

1/64

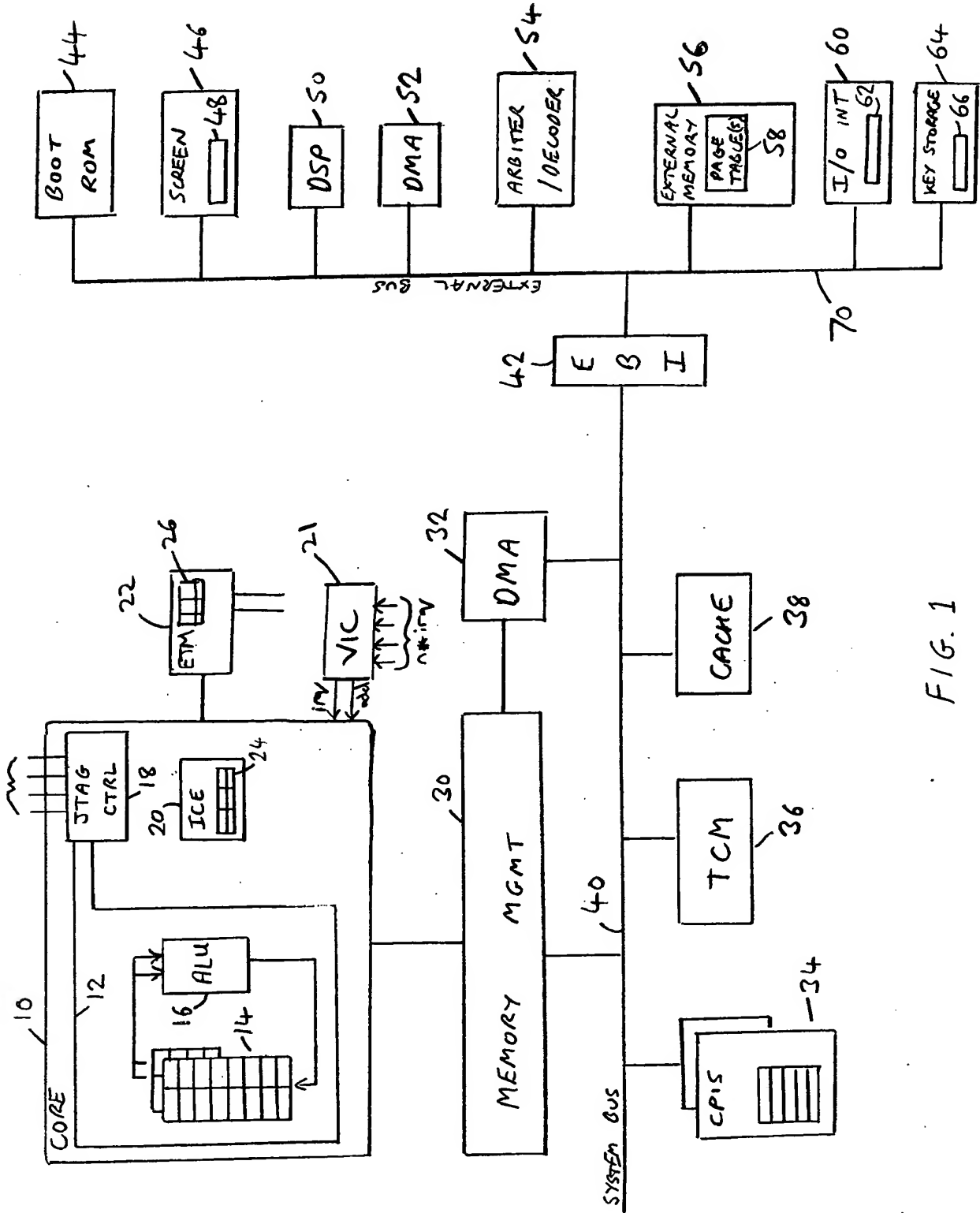


FIG. 1

2/64

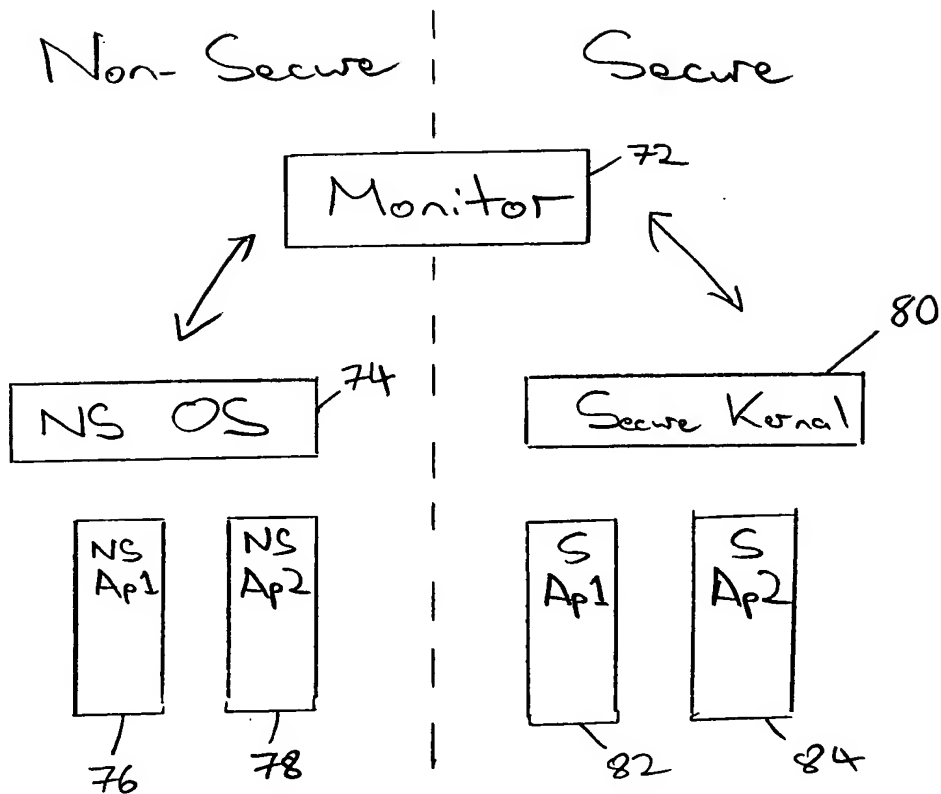


Fig. 2

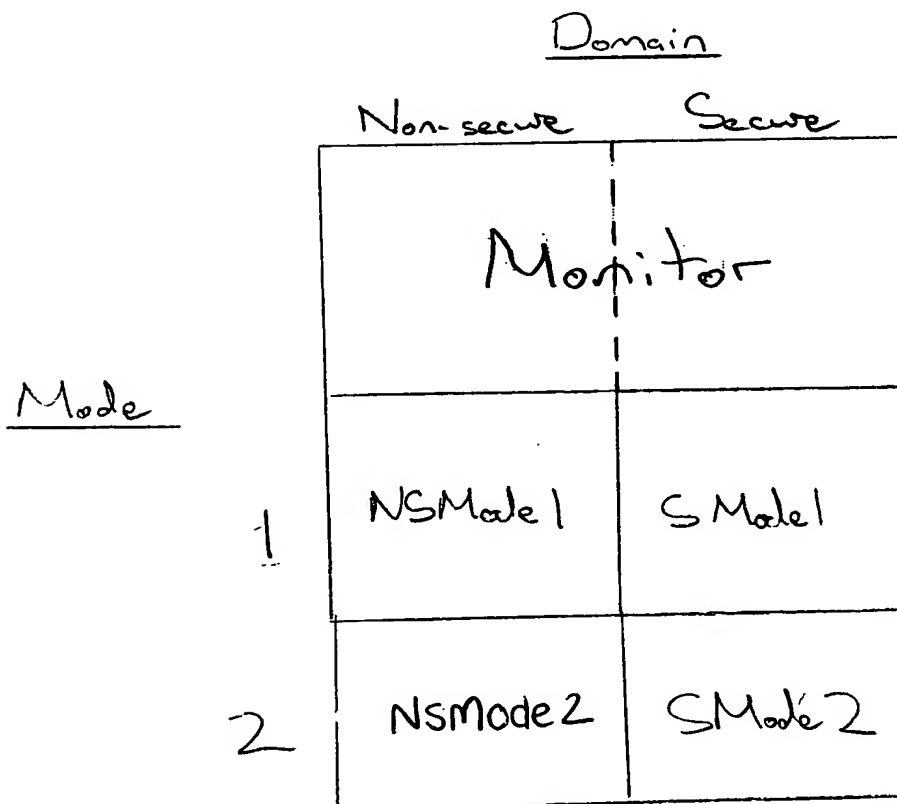


Fig. 3

3/64

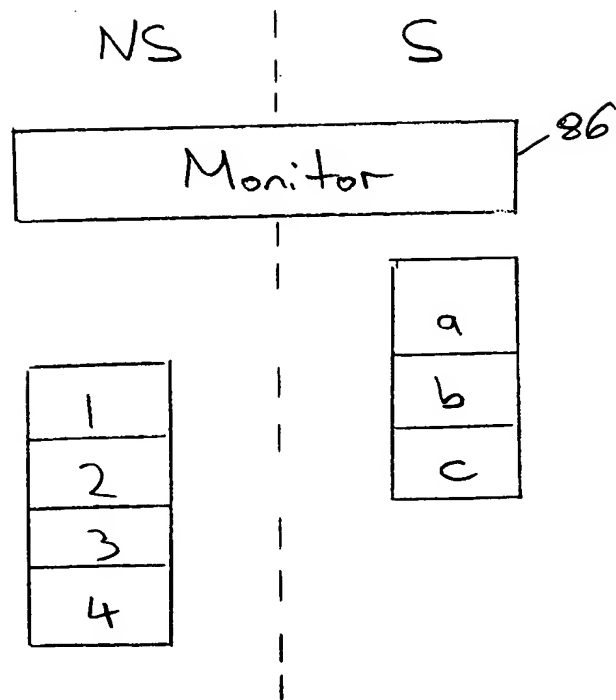


Fig. 4

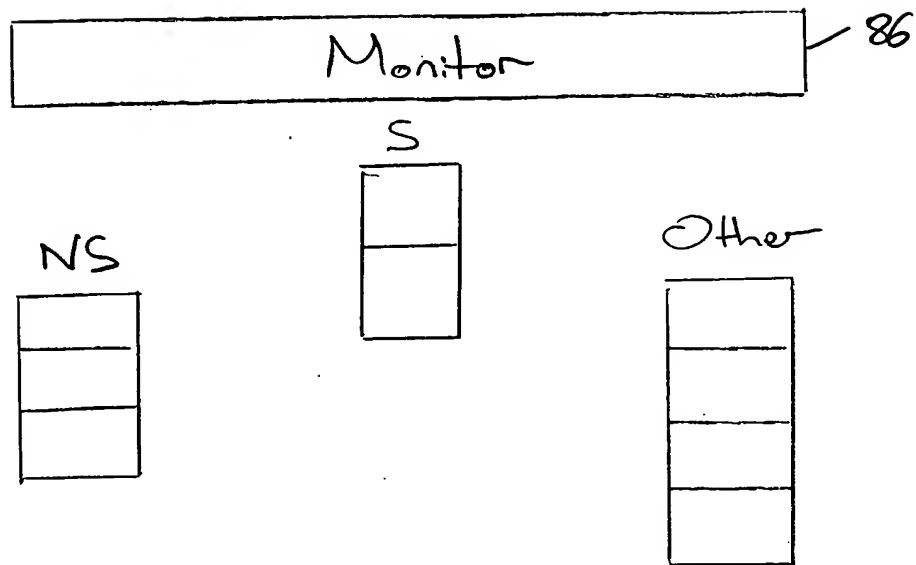
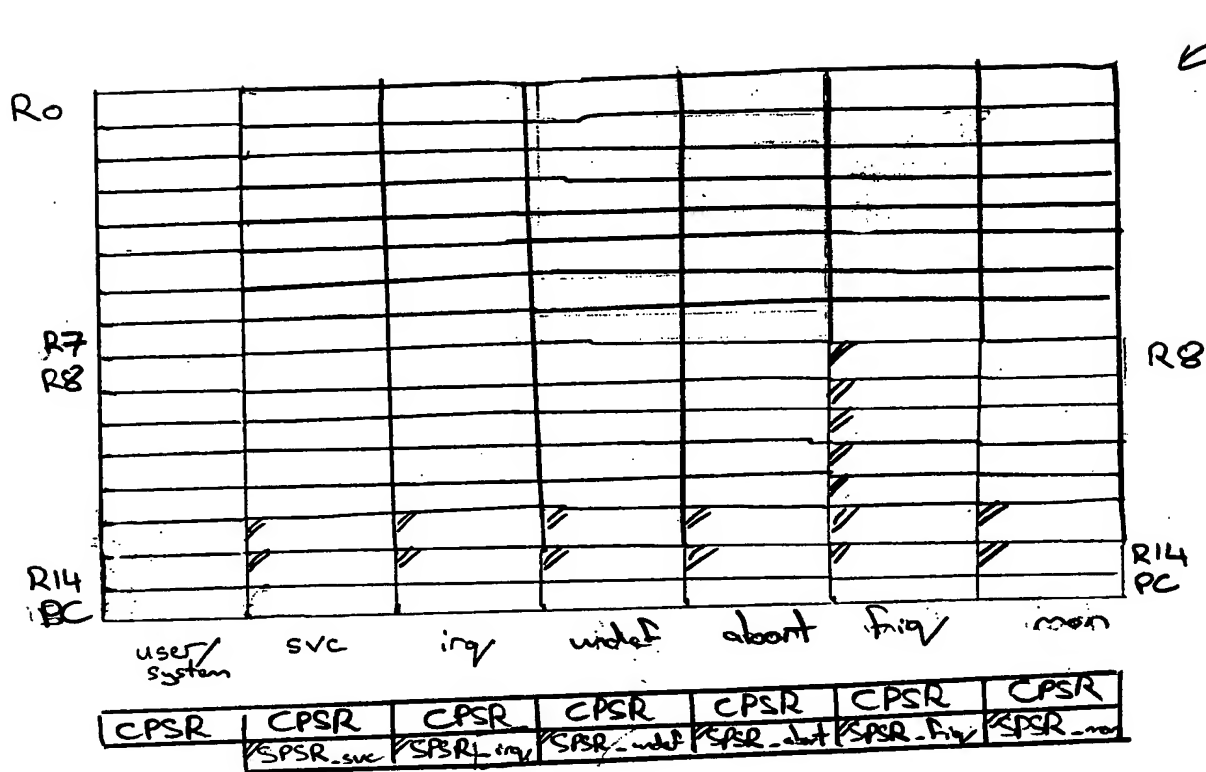


Fig. 5

4/64



// = private to mode

fig. 6

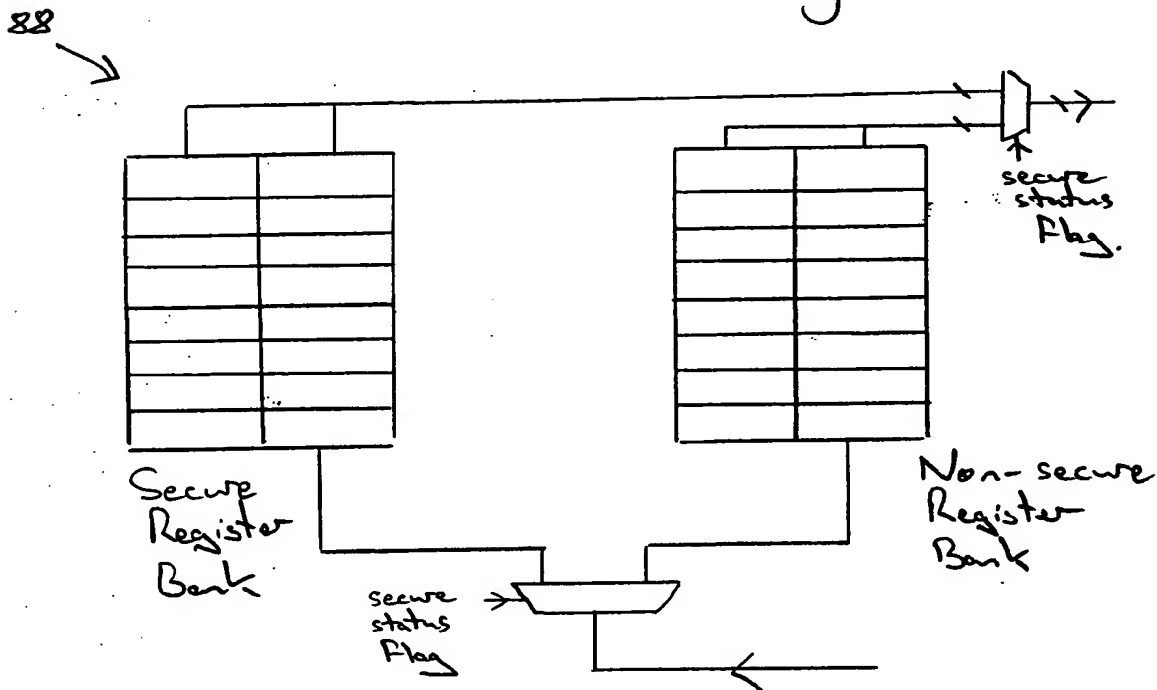


Fig. 7

5/64

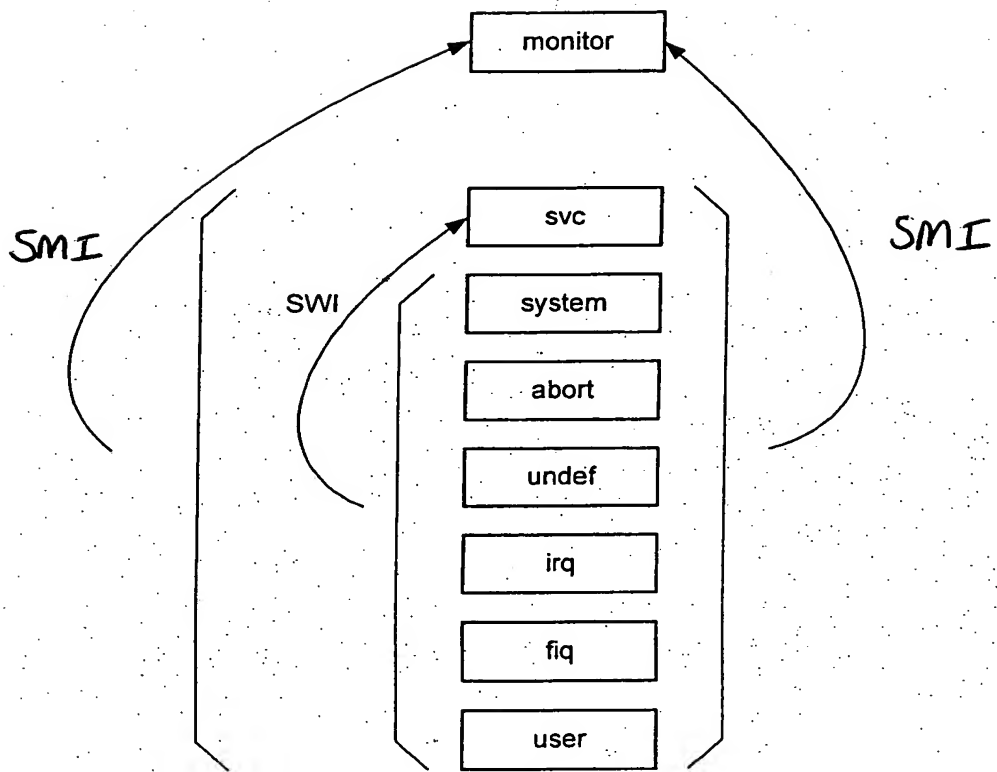


Fig. 8

6/64

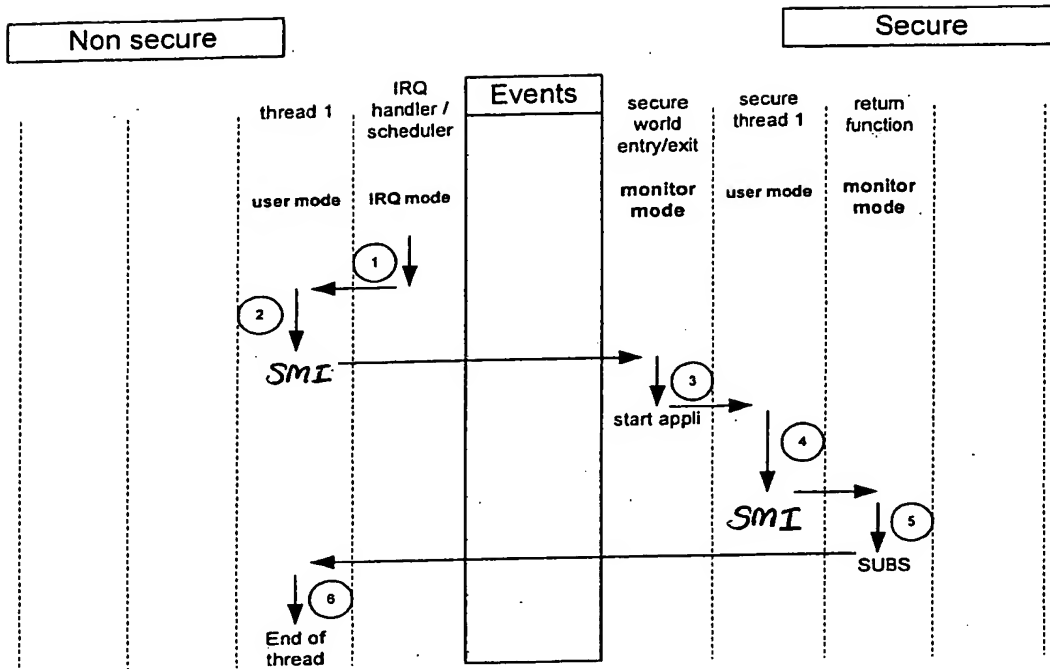


Fig. 9

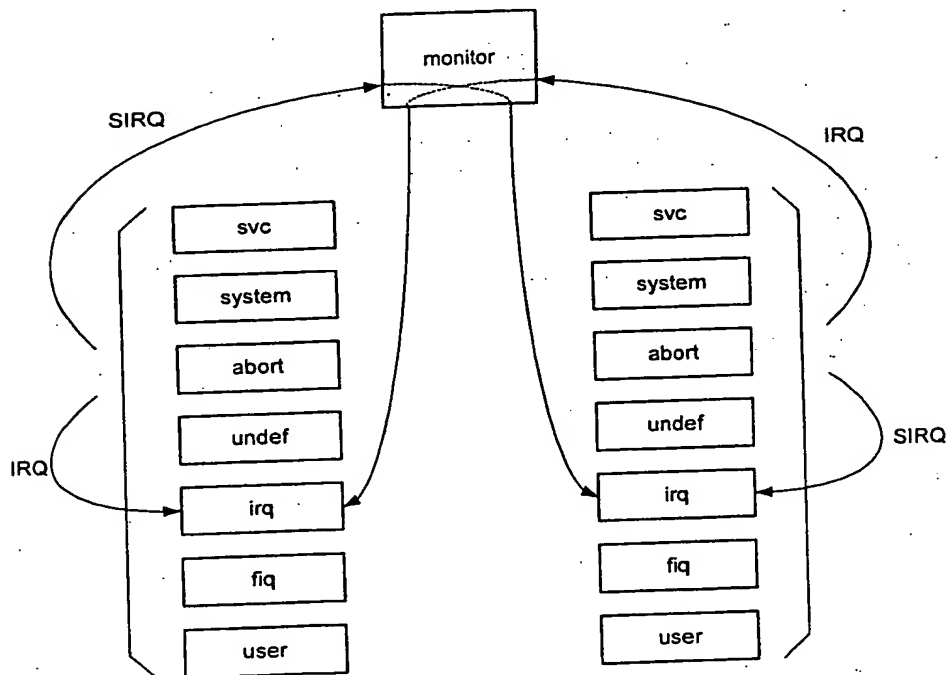


Fig. 10

7/64

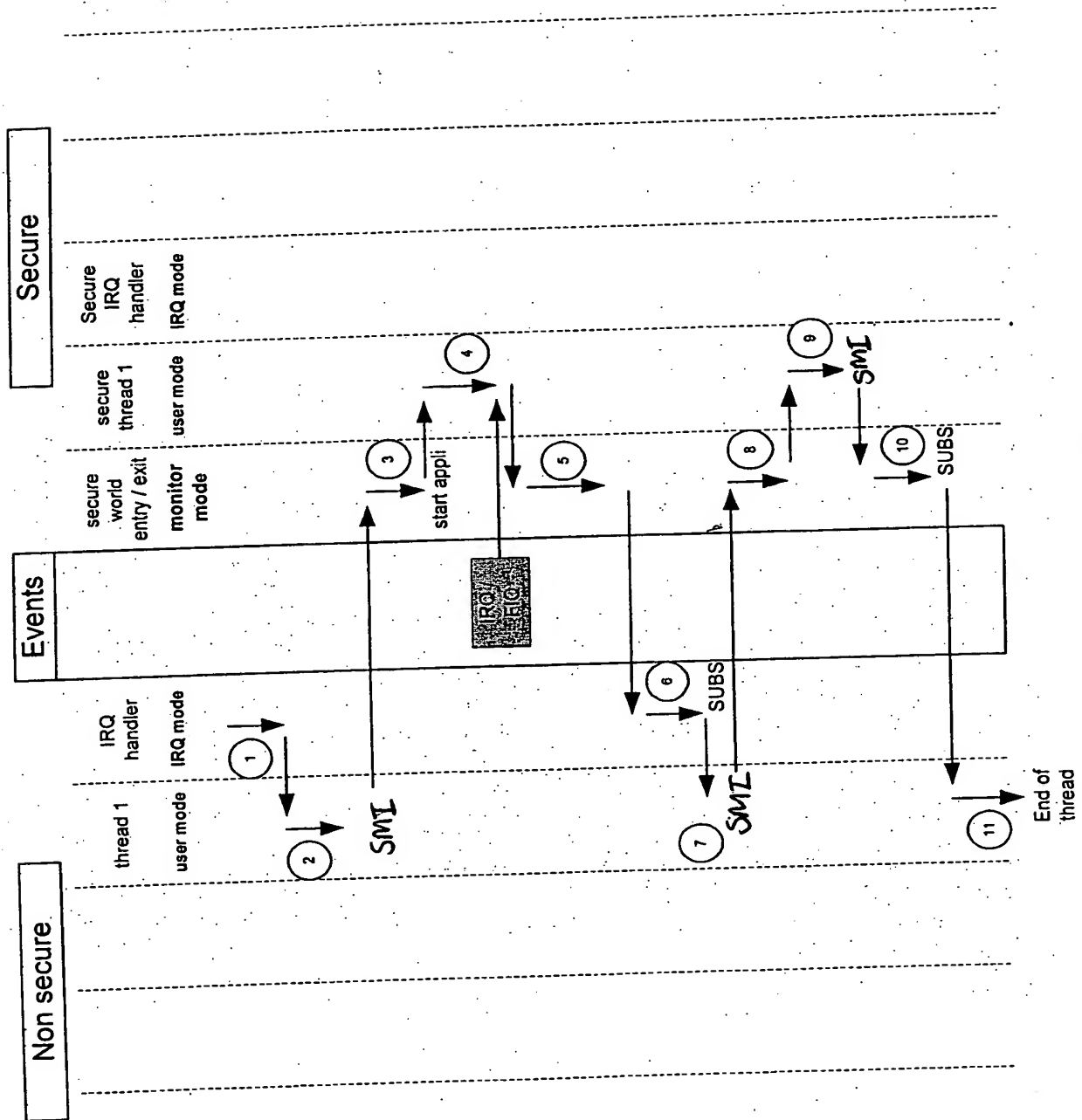


Fig. 11A





```

sequenceDiagram
    participant NS as Non secure
    participant E as Events
    participant S as Secure

    Note over NS: thread 1  
user mode
    Note over S: Secure IRQ handler  
IRQ mode

    NS-->>E: (1)
    E-->>NS: (2)
    Note over NS: SMI
    NS-->>S: (3)
    Note over S: start appli
    S-->>E: (4)
    Note over E: IRQ/FIQ
    E-->>S: (5)
    Note over S: SMI
    S-->>E: (6)
    E-->>NS: (7)
    Note over NS: SUBS
    NS-->>S: (8)
    Note over S: SMI
    S-->>E: (9)
    Note over E: SUBS
    E-->>NS: (10)
    Note over NS: End of thread
  
```

Fig. 13A

10/64

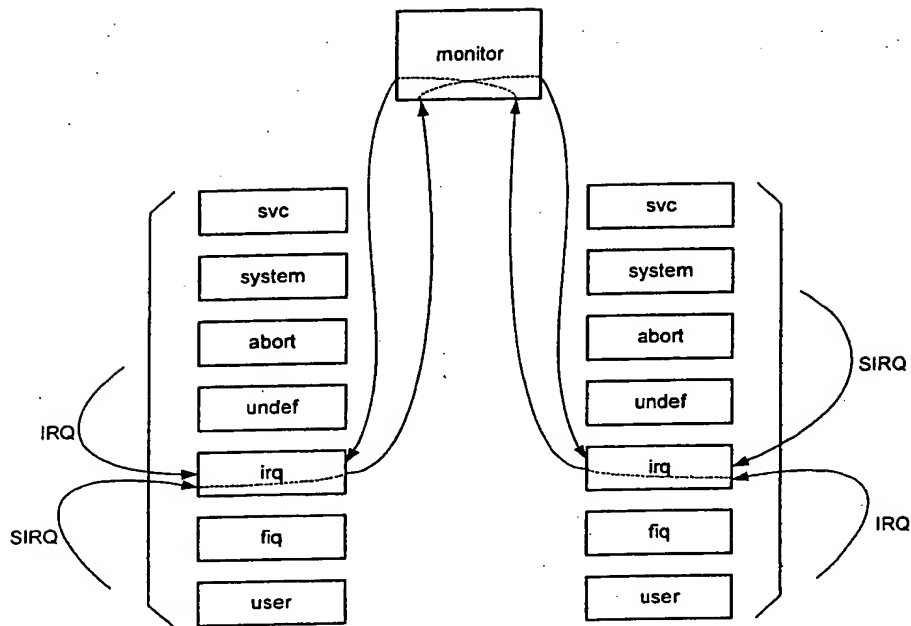


Fig. 12

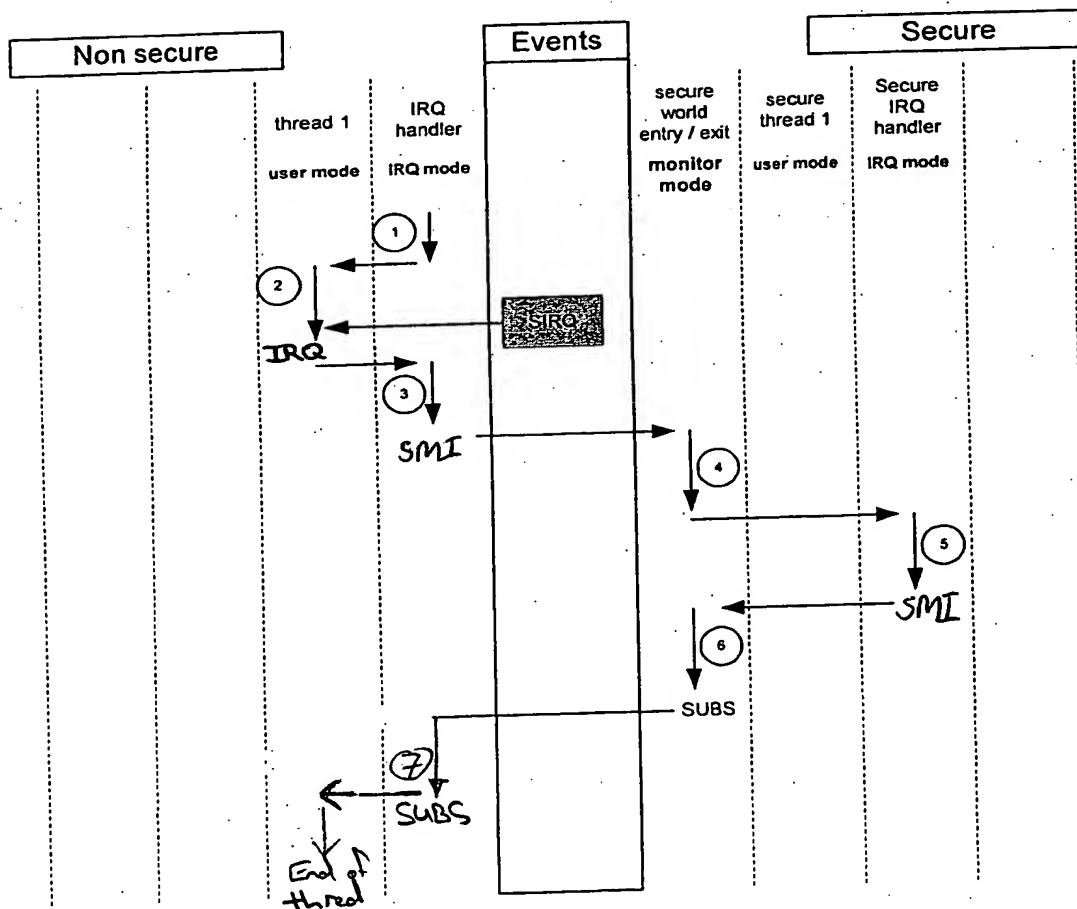


Fig. 13B

11/64

Exception	Vector offset	Corresponding mode
Reset	0x00	Supervisor mode
Undef	0x04	Monitor mode / Undef mode
SWI	0x08	Supervisor mode / Monitor mode
Prefetch abort	0x0C	Abort mode / Monitor mode
Data abort	0x10	Abort mode / Monitor mode
IRQ / SIRQ	0x18	IRQ mode / Monitor mode
FIQ	0x1C	FIQ mode / Monitor mode
SMT	0x20	Undef mode / Monitor mode

Fig. 14

Monitor	
Reset	VM0
Undef	VM1
SWI	VM2
Prefetch abort	VM3
Data abort	VM4
IRQ / SIRQ	VM5
FIQ	VM6
SMT	VM7

Secure

Non-Secure

Reset	VS0
Undef	VS1
SWI	VS2
Prefetch abort	VS3
Data abort	VS4
IRQ / SIRQ	VS5
FIQ	VS6
SMT	VS7

Reset	VNS0
Undef	VNS1
SWI	VNS2
Prefetch abort	VNS3
Data abort	VNS4
IRQ / SIRQ	VNS5
FIQ	VNS6
SMT	VNS7

Fig. 15

CPI5 Monitor Trap Mask Register

only NS world exceptions illustrated

0	1	1	1	1	0	1
SMT	SWI	Prefetch Abort	Data Abort	IRQ	SIRQ	FIQ



OR via hardware/external pin

1 = Mon(S)

0 = NS

Fig. 16

12/64

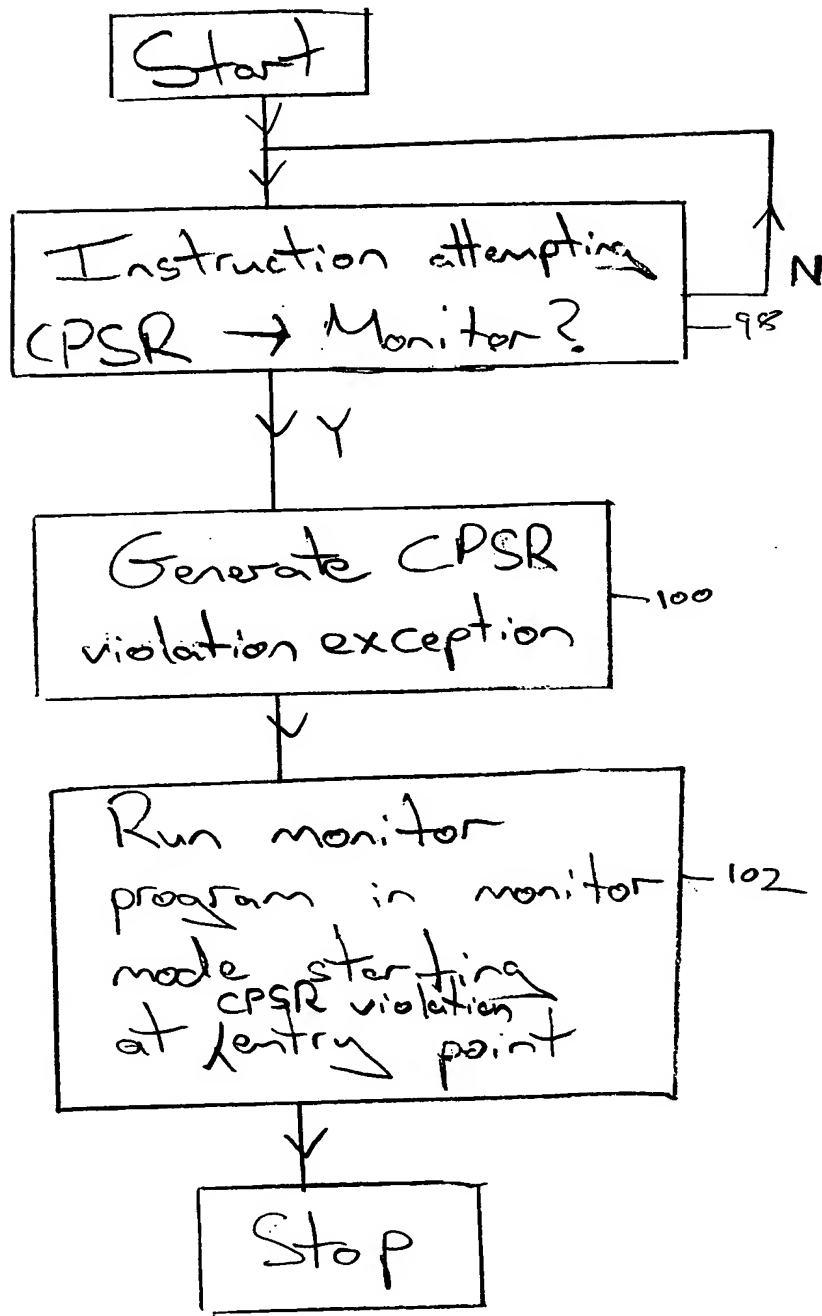


Fig. 17

13/64

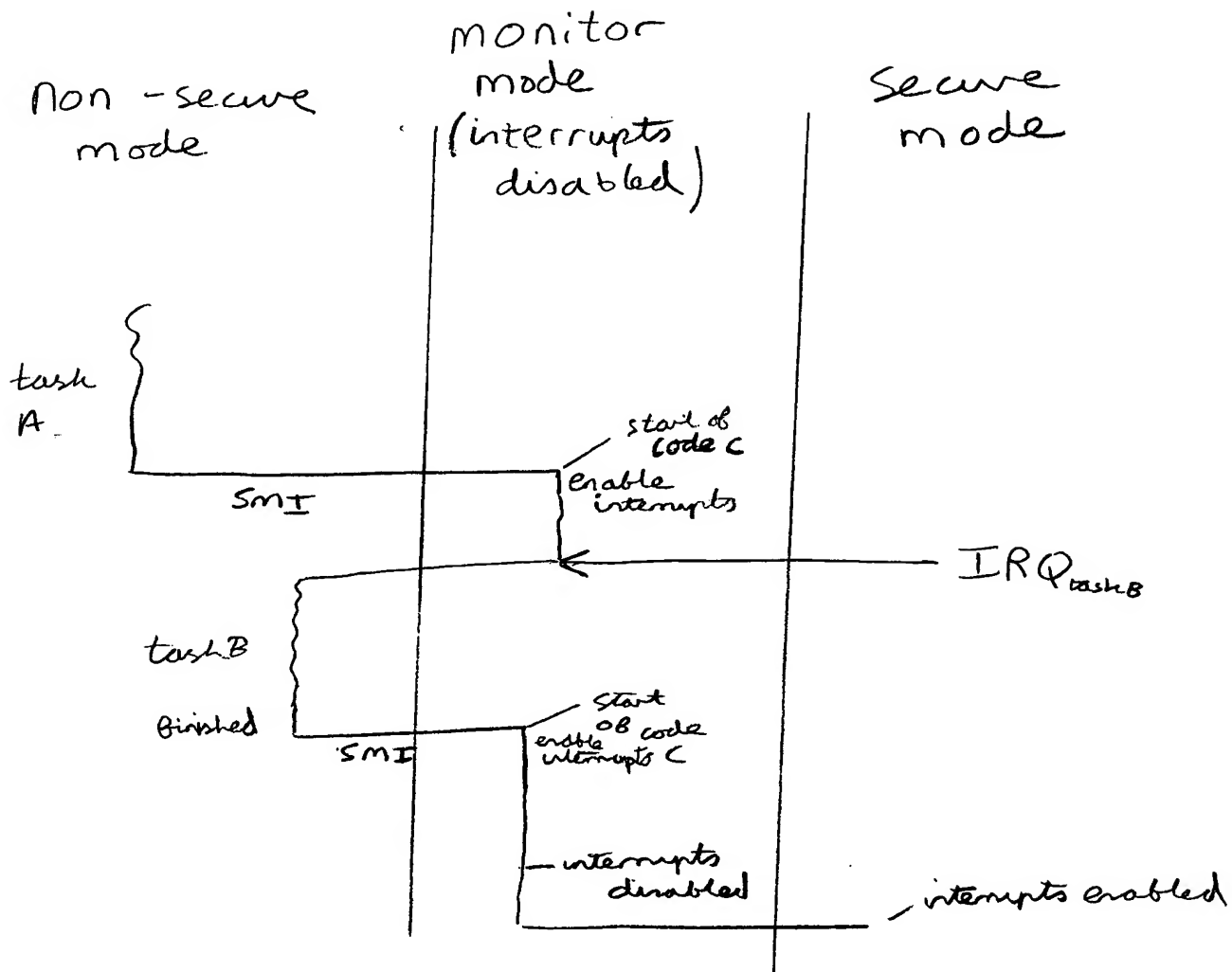


Fig. 18



15/64

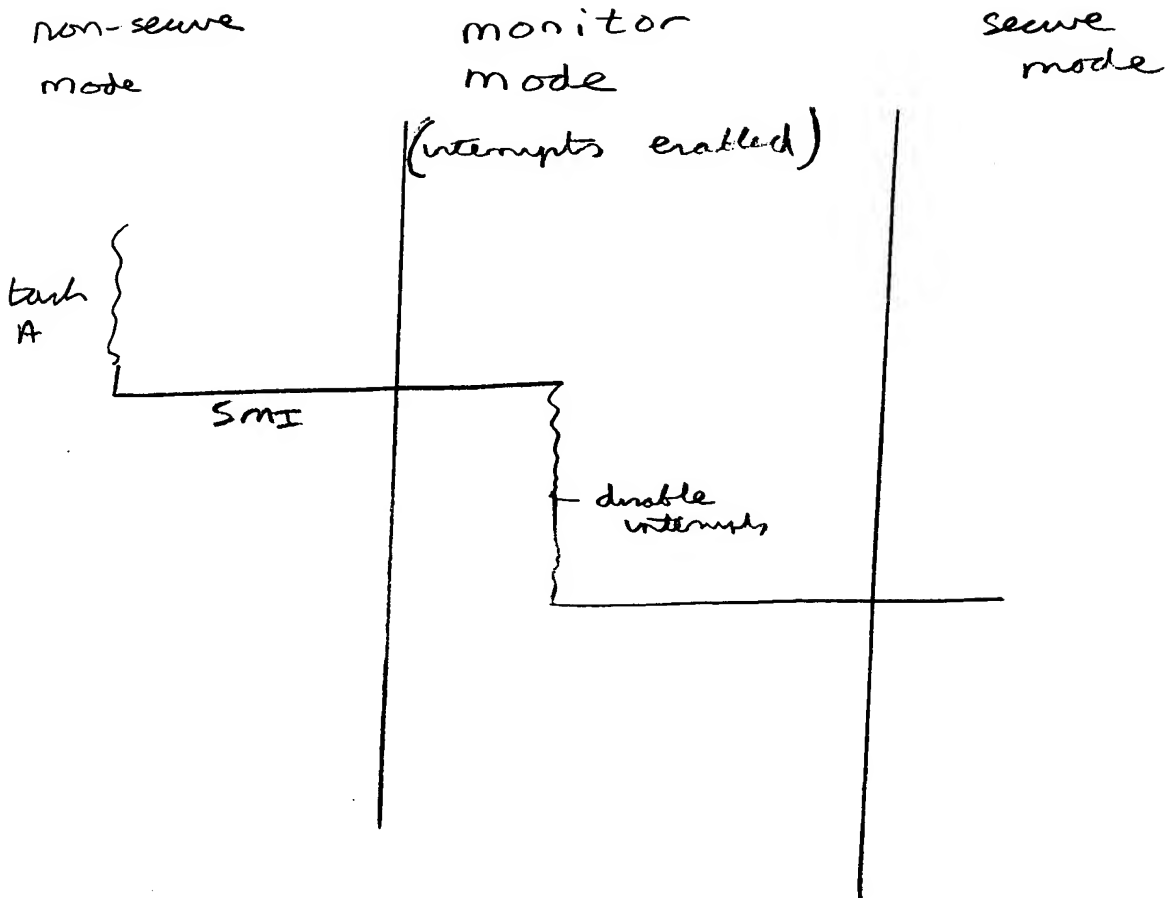


Fig. 20

16/64

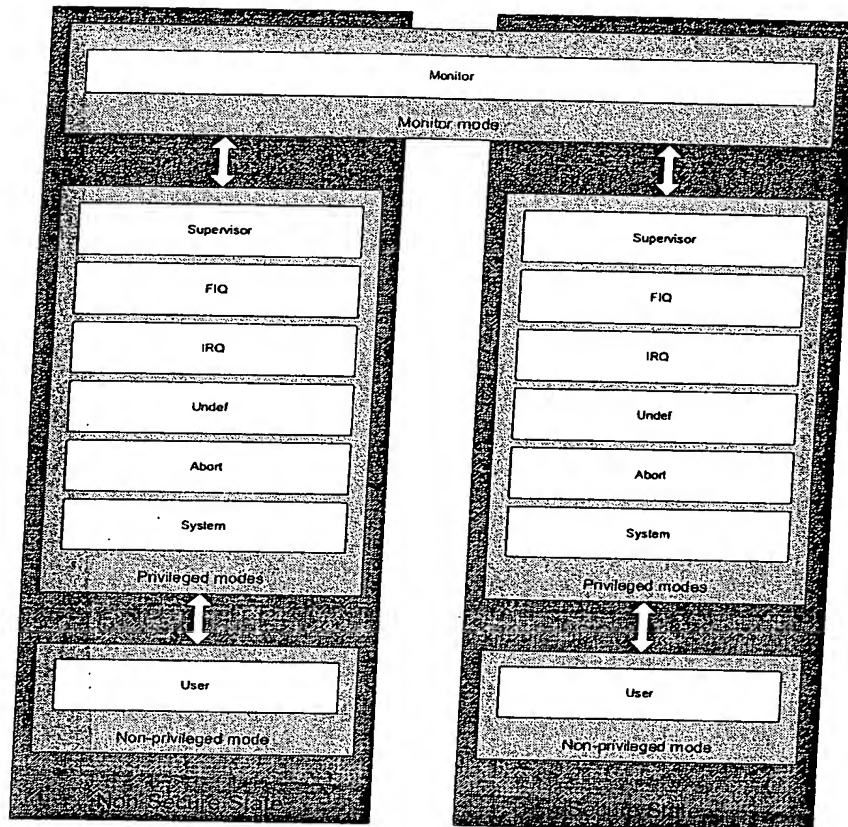


FIGURE 21



17/64

User	System	Supervisor	Abort	Undefined	Interrupt	Fast Interrupt	Monitor
R0	R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq	R8
R9	R9	R9	R9	R9	R9	R9_fiq	R9
R10	R10	R10	R10	R10	R10	R10_fiq	R10
R11	R11	R11	R11	R11	R11	R11_fiq	R11
R12	R12	R12	R12	R12	R12	R12_fiq	R12
R13	R13	<del>R13_svc</del>	<del>R13_abt</del>	R13_und	R13_irq	R13_fiq	<del>R13_mon</del>
R14	R14	<del>R14_svc</del>	<del>R14_sbt</del>	R14_und	R14_irq	R14_fiq	<del>R14_mon</del>
PC	PC	PC	PC	PC	PC	PC	PC

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq	SPSR_mon

FIGURE 22

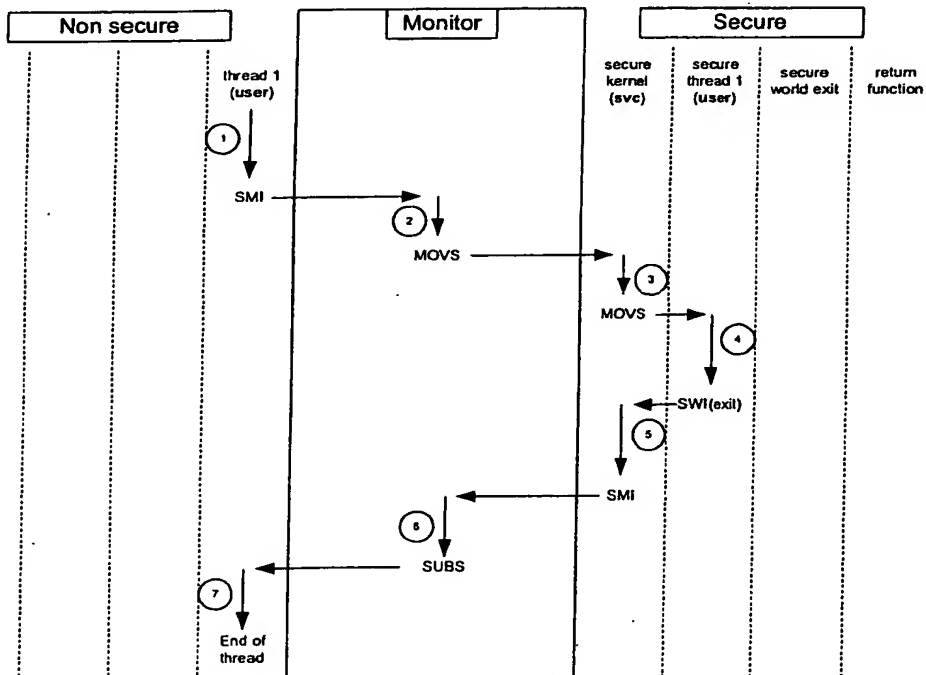
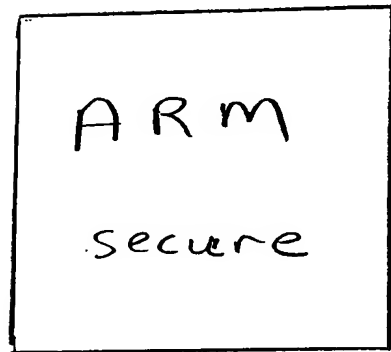
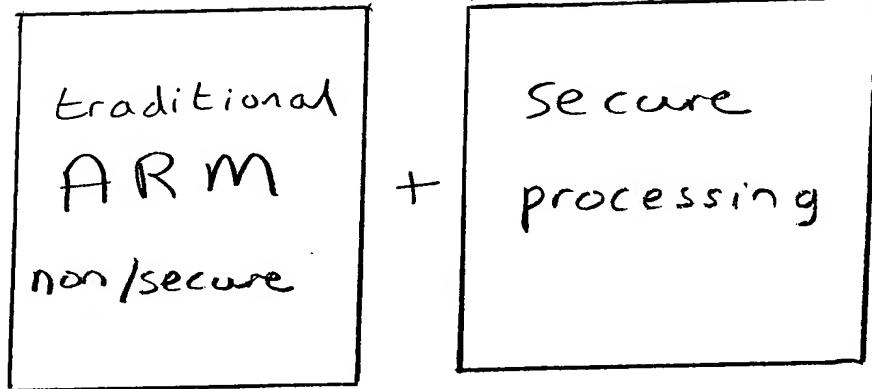


FIGURE 23

18/64



$S = 1$

Fig. 24

19/64

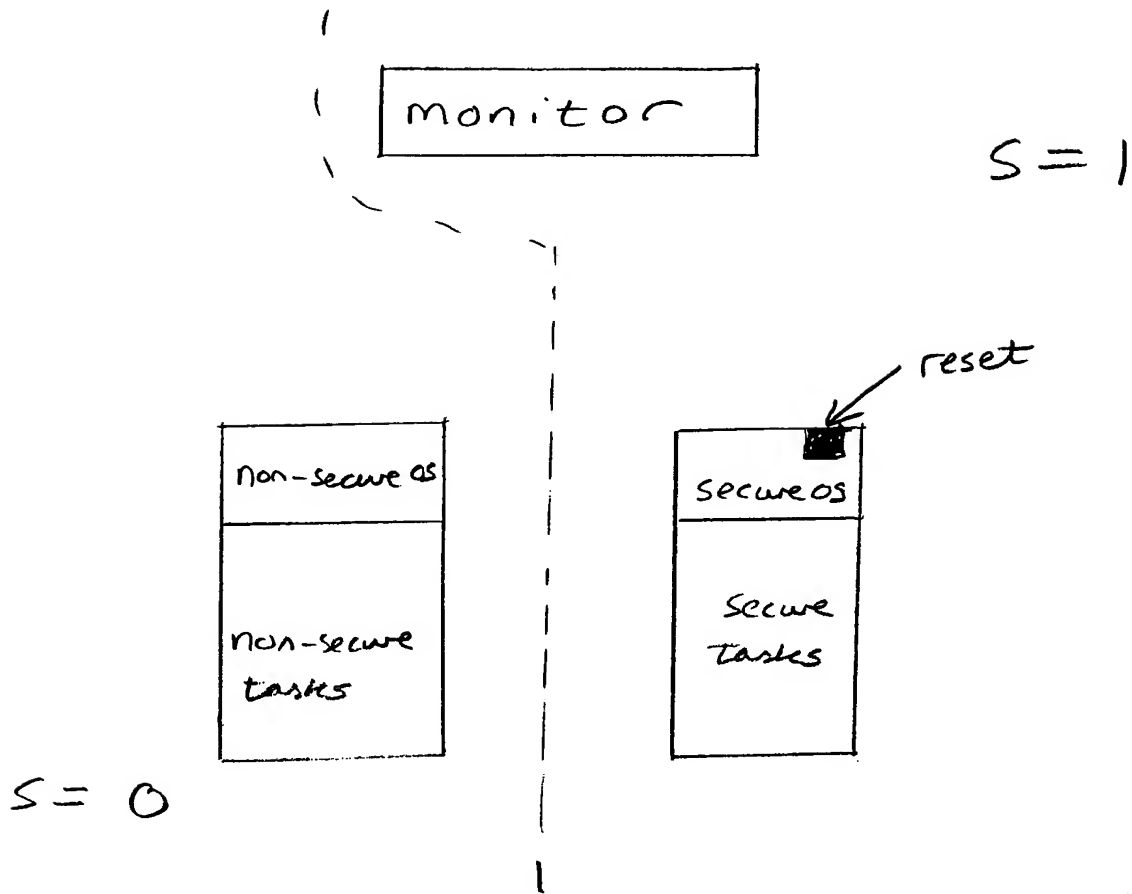


Fig. 25

20/64

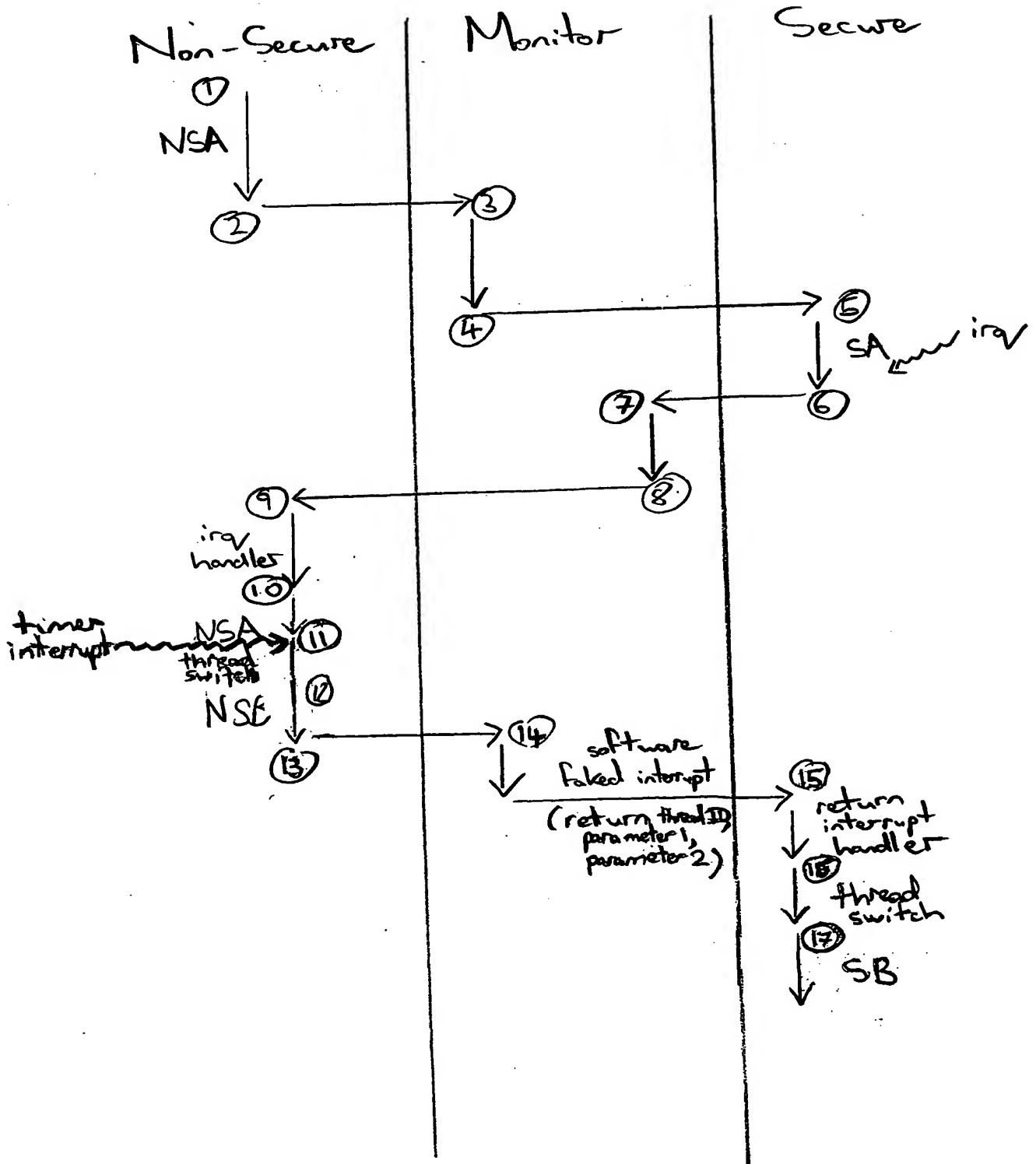


fig. 26

21/64

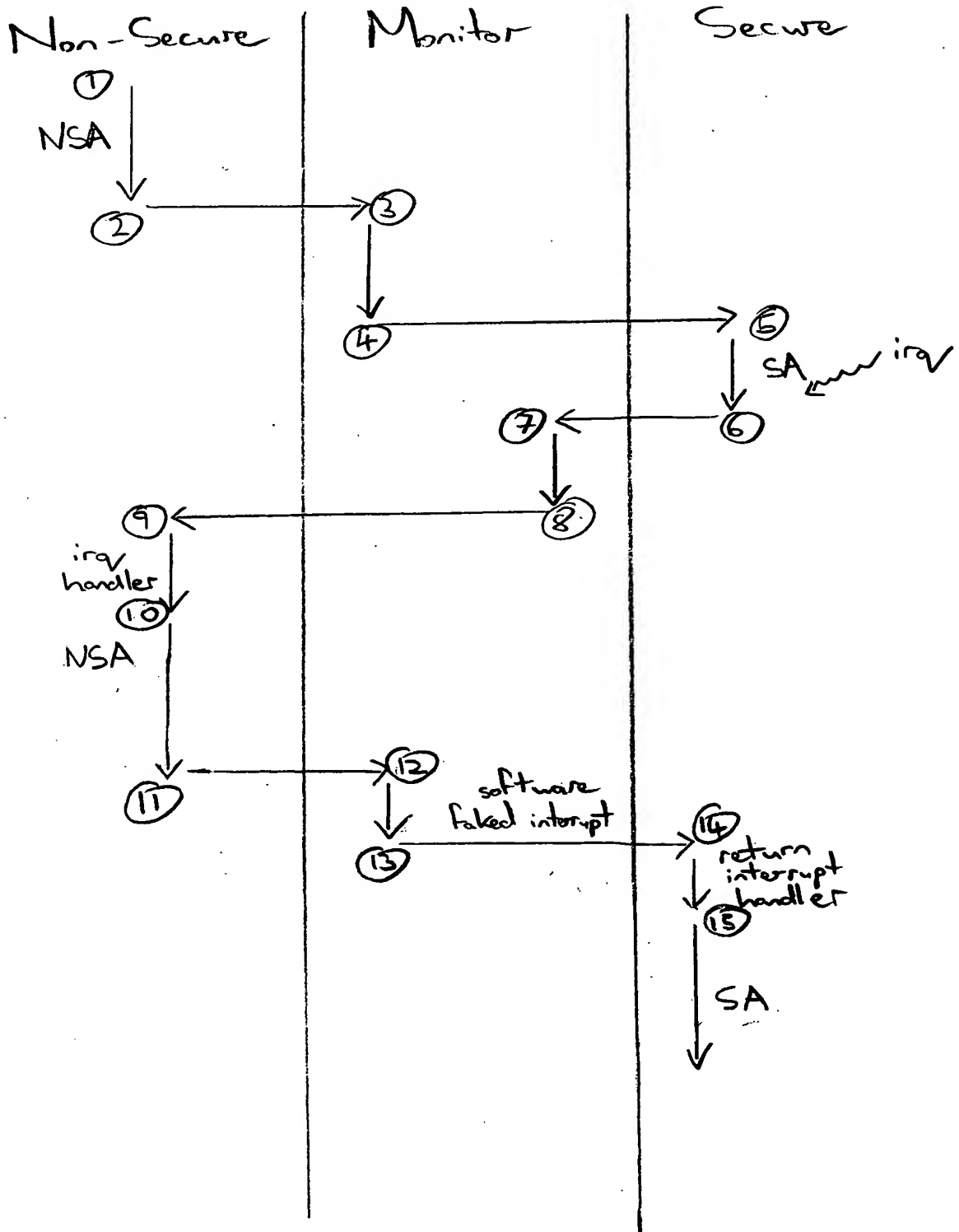


Fig. 27

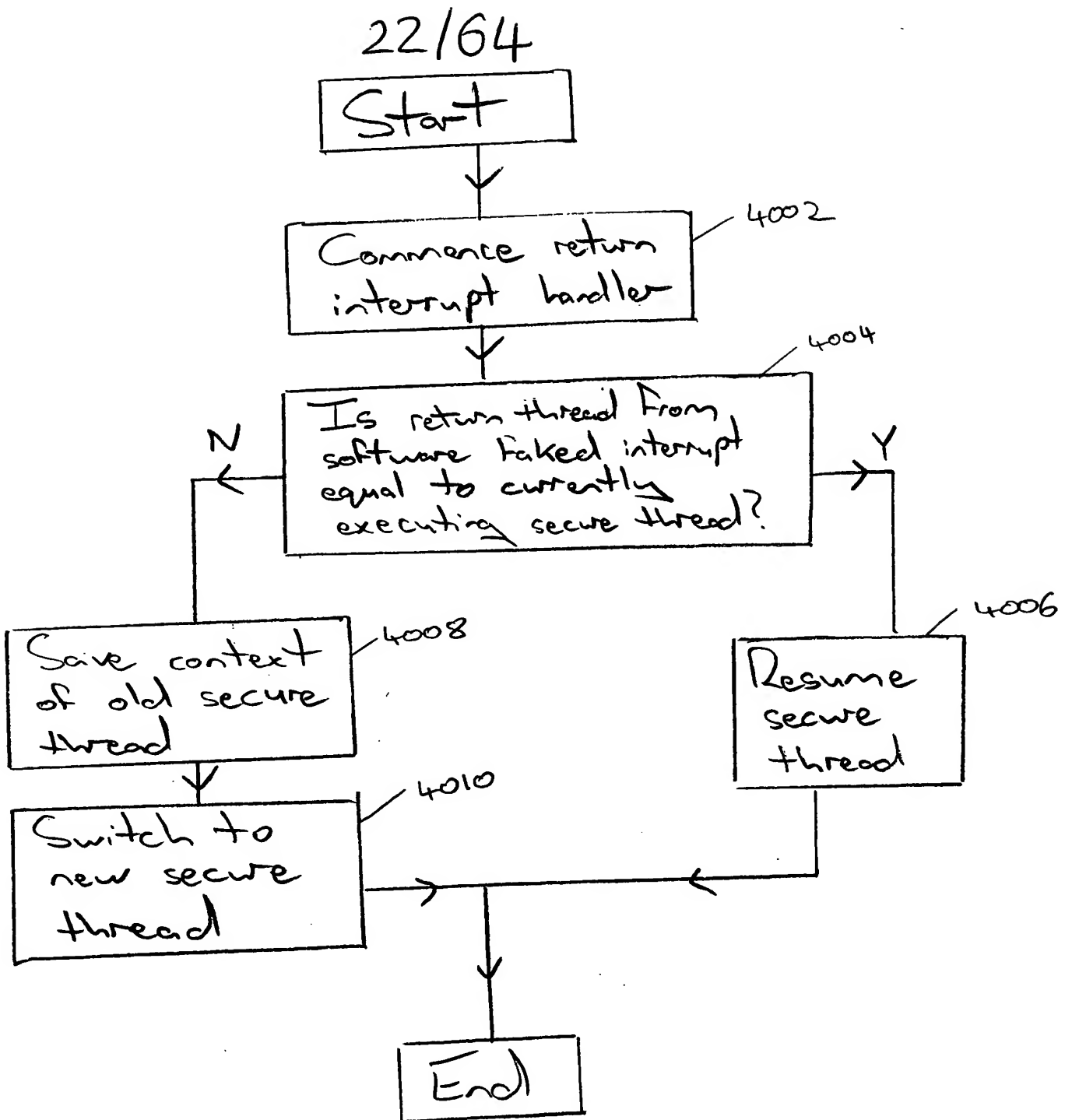


Fig. 28

23/64

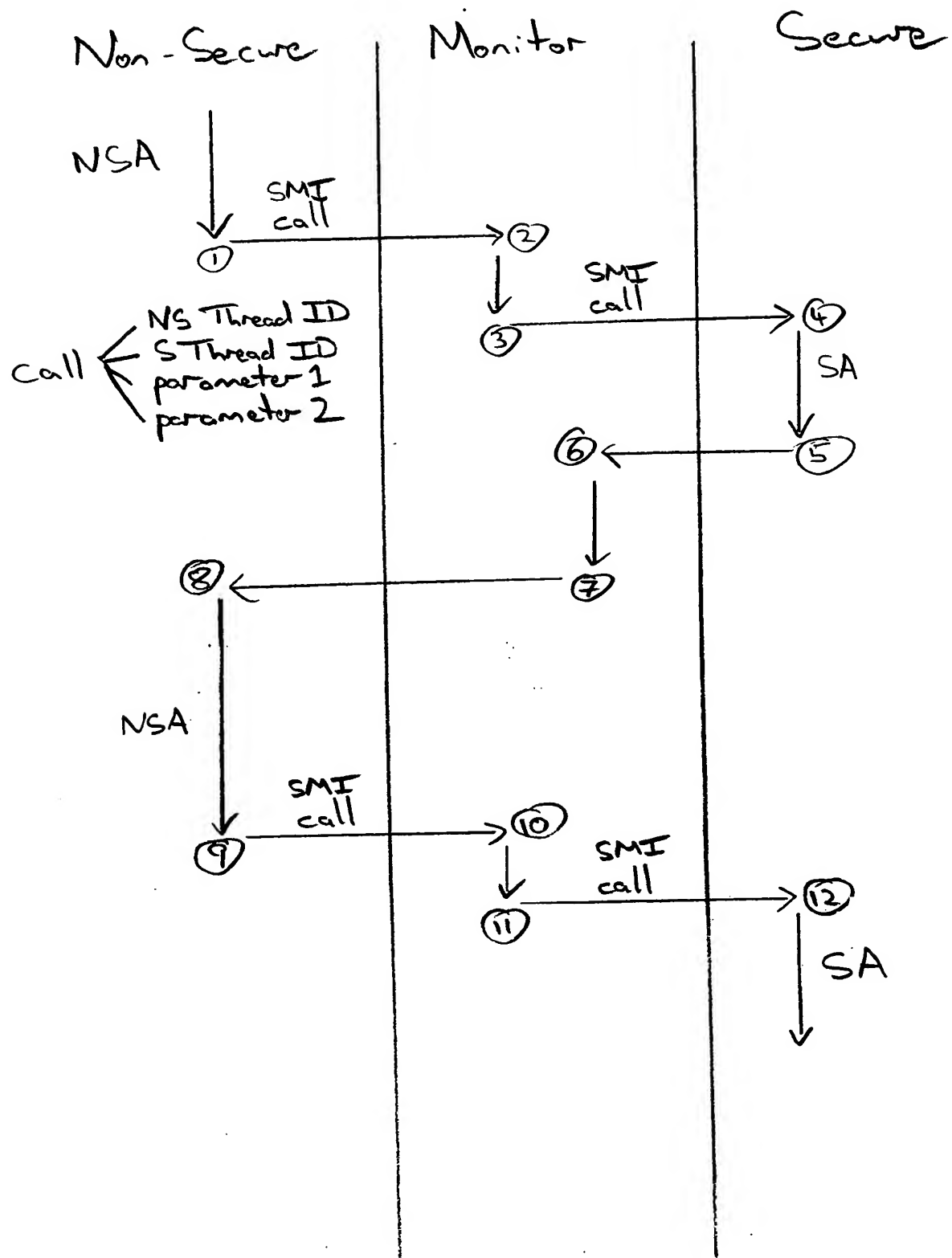


Fig. 29

24/64

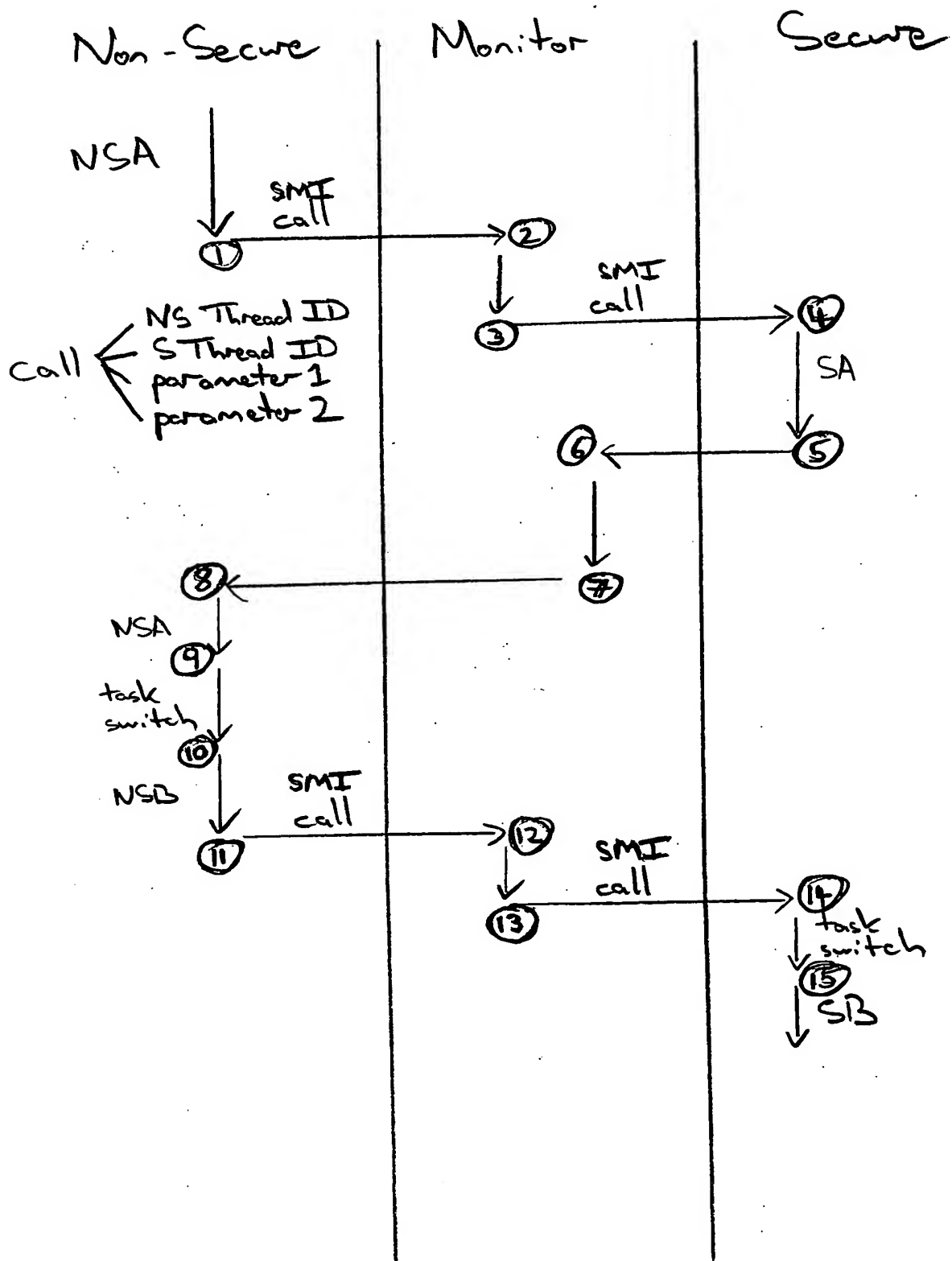


Fig. 30



25/64

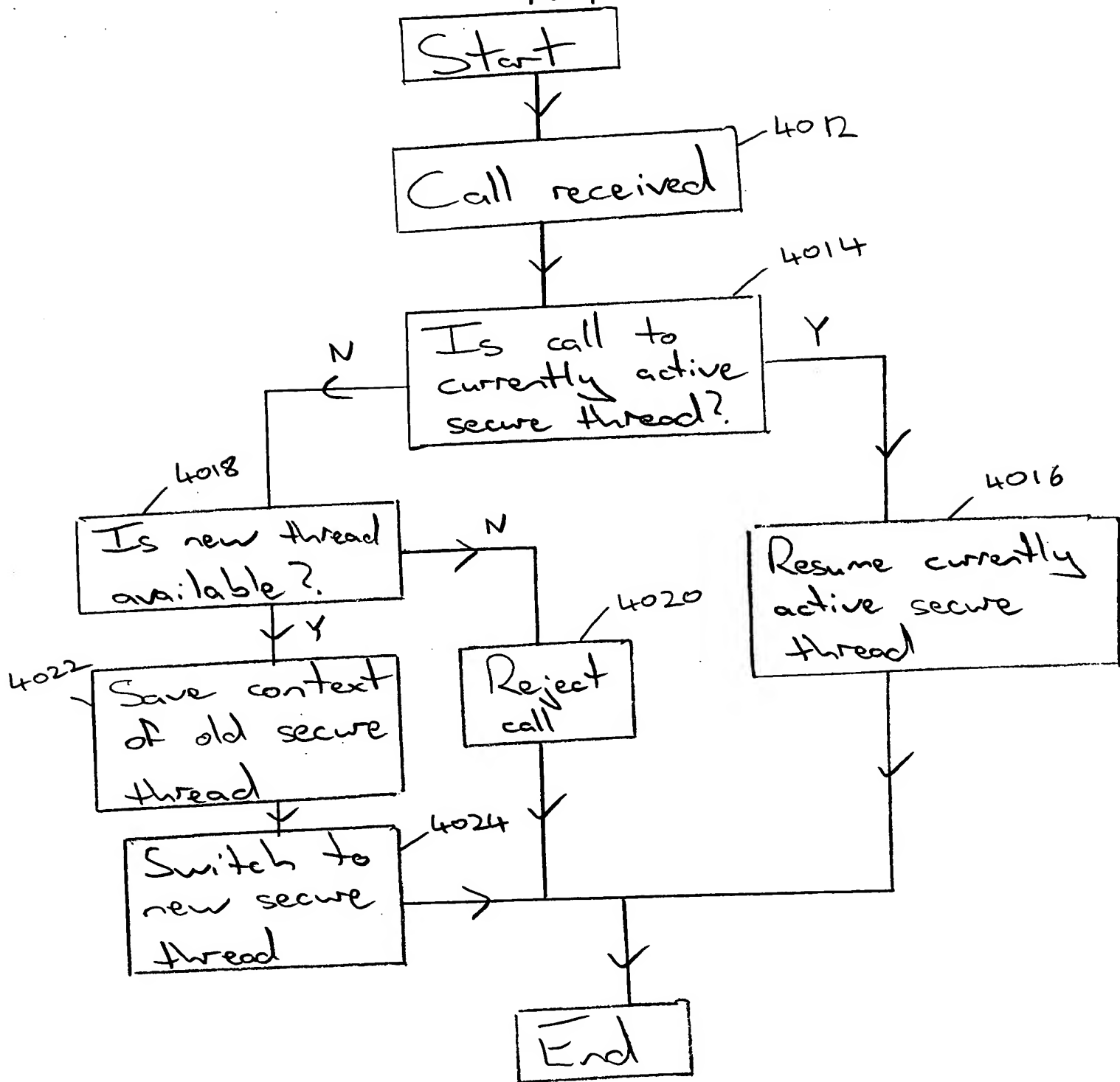


Fig. 31



27/64

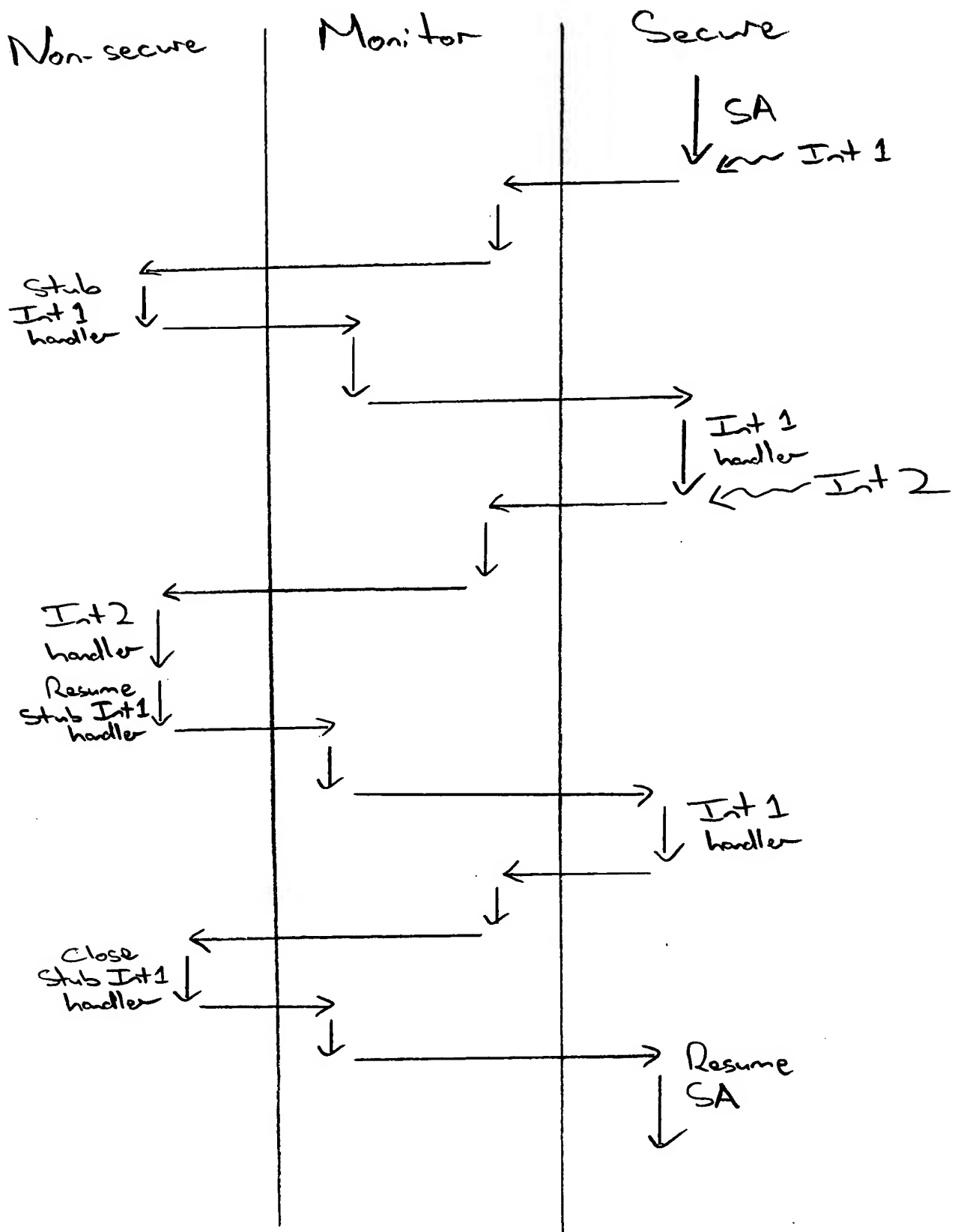


Fig 33

28/64

<u>Interrupt Type / Priority</u>	<u>How Handled</u>	
1	S	
2	S	
3	NS	
4	NS/S	no S only handlers lower than highest NS handler
5	NS	
6	NS/S	
7	NS	
⋮	⋮	

Fig. 34

29/64

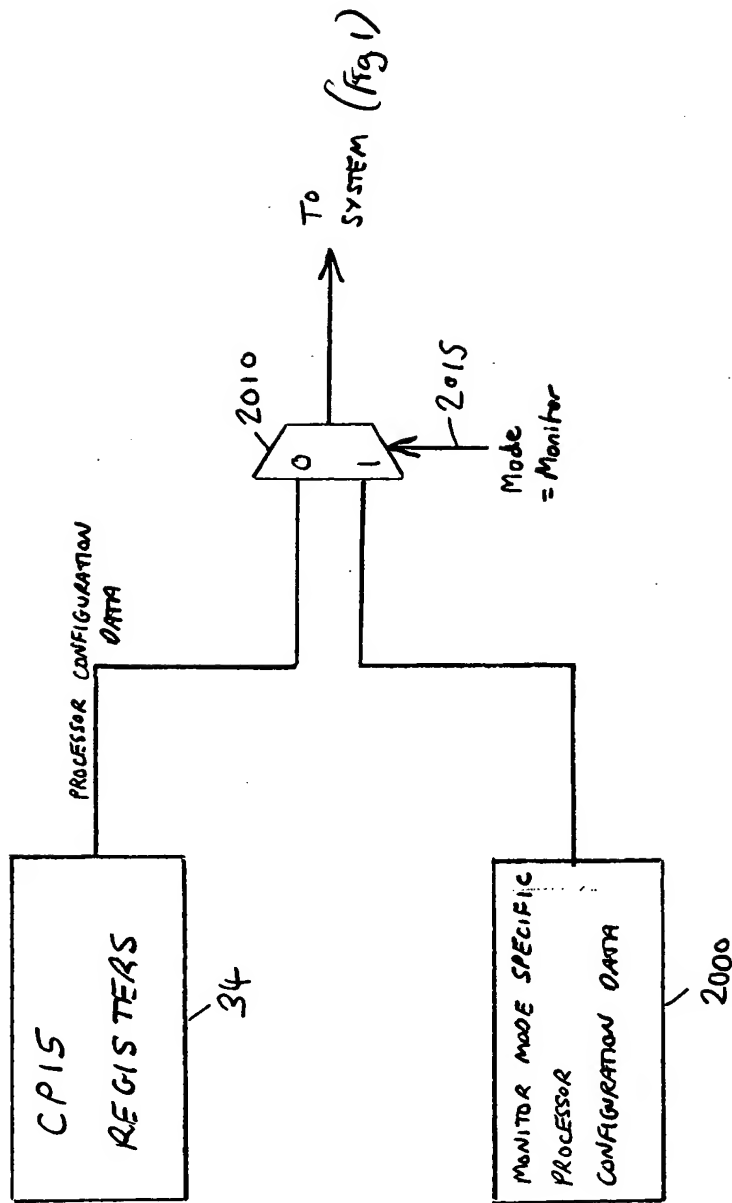


FIG. 35

30/64

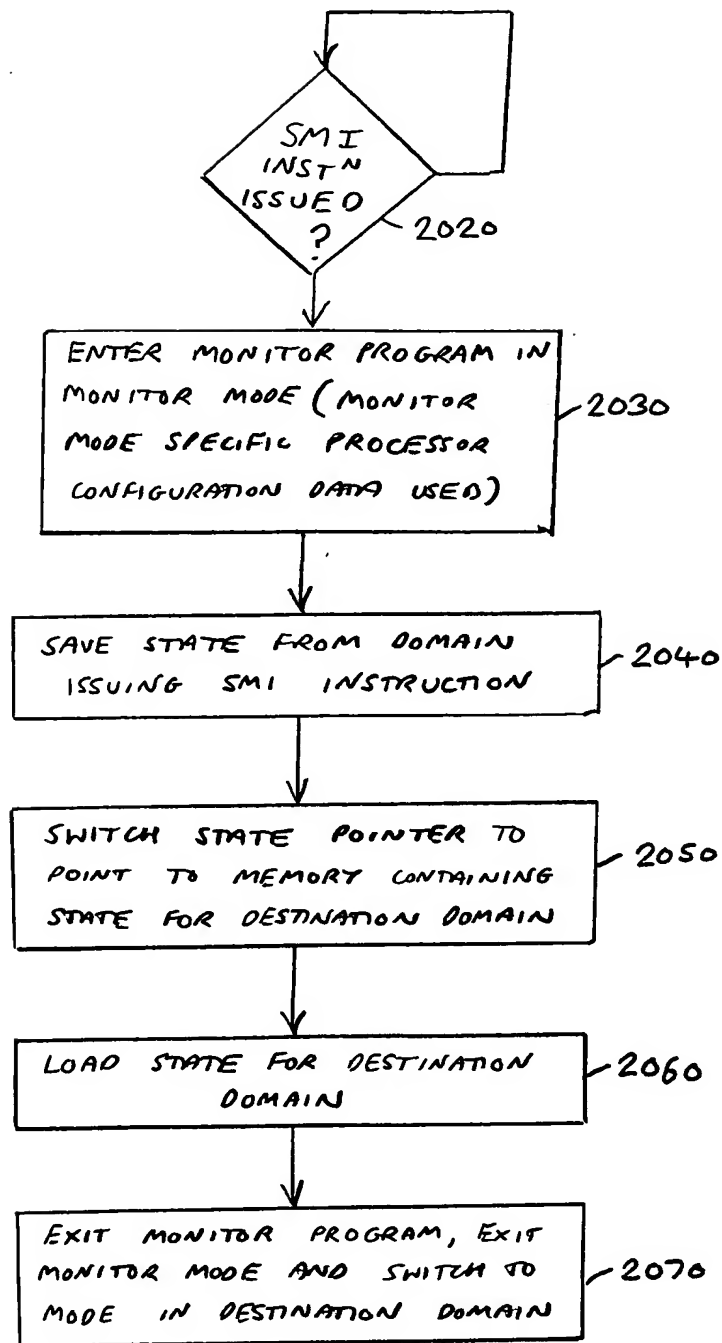


FIG. 36

31/64

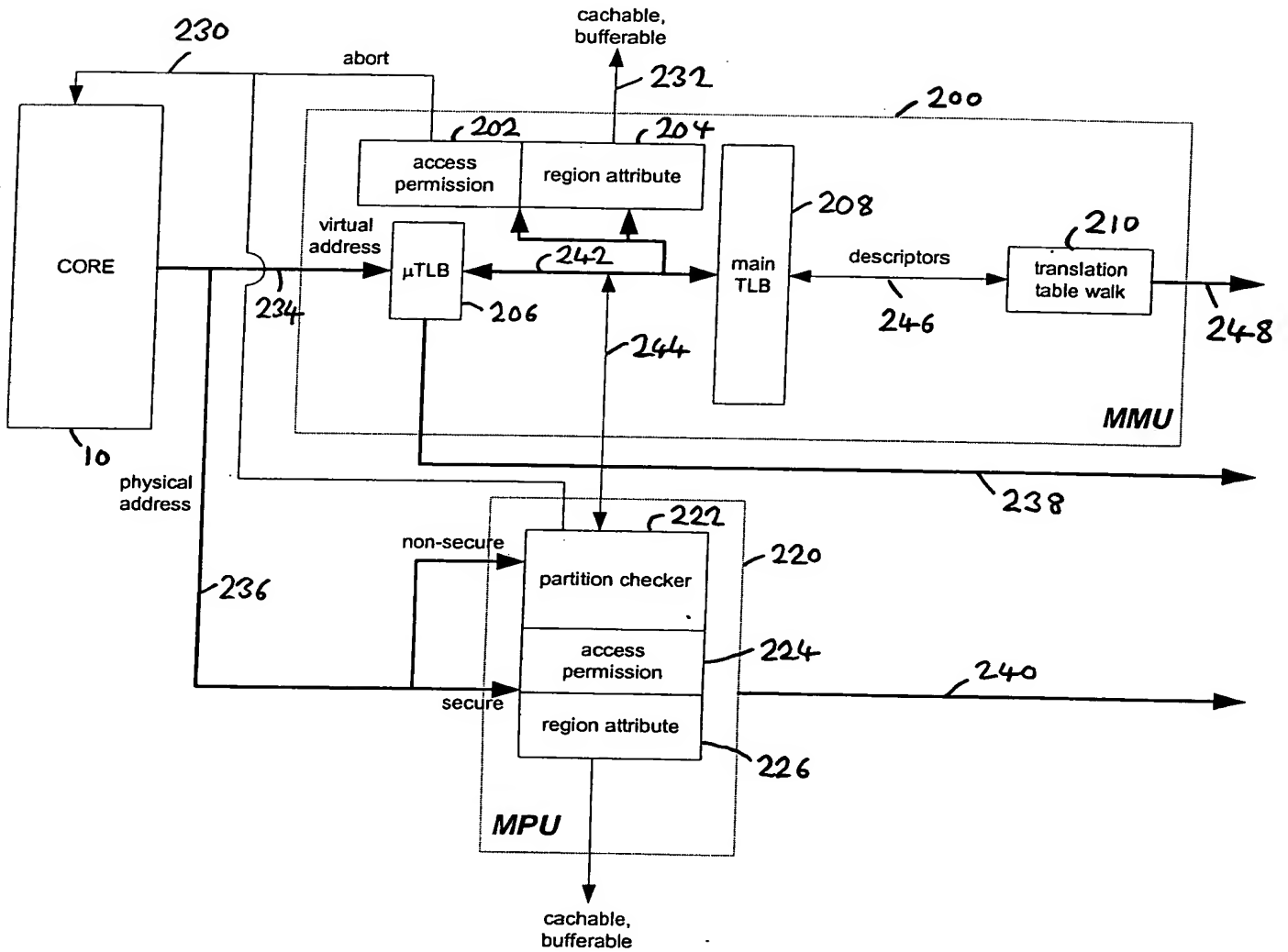


FIG. 37

32/64

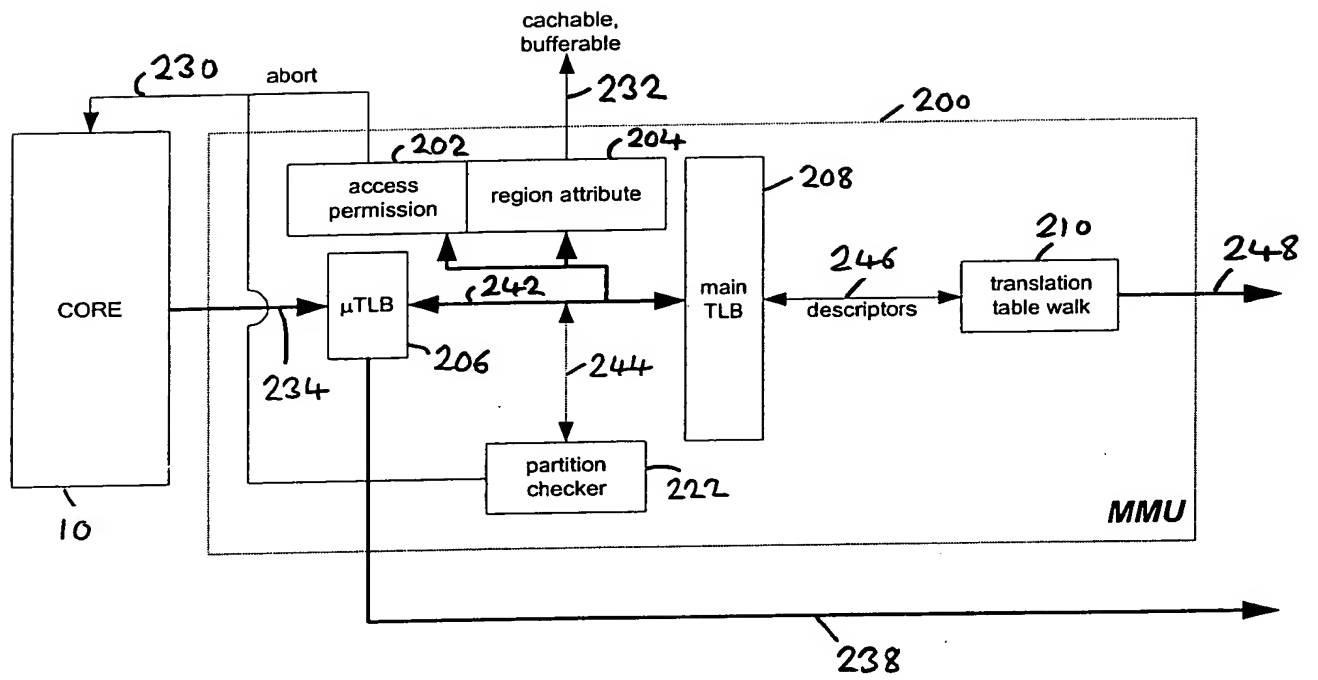


FIG. 38



33/64

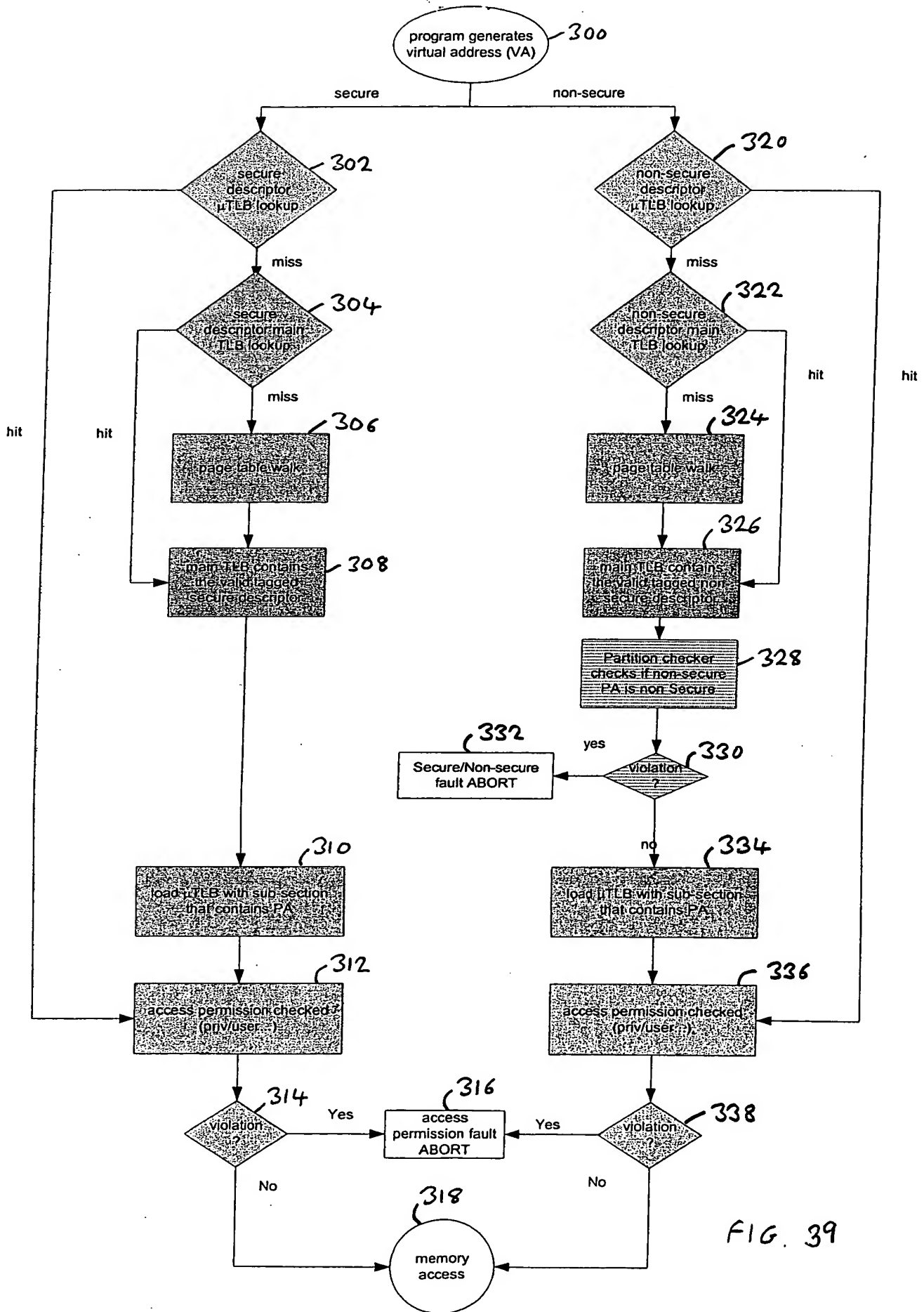


FIG. 39

34/64

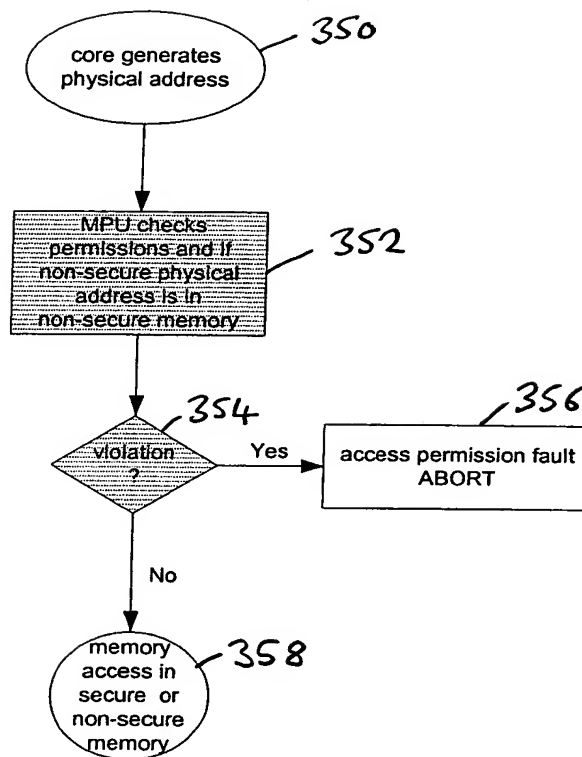


FIG. 40

35/64

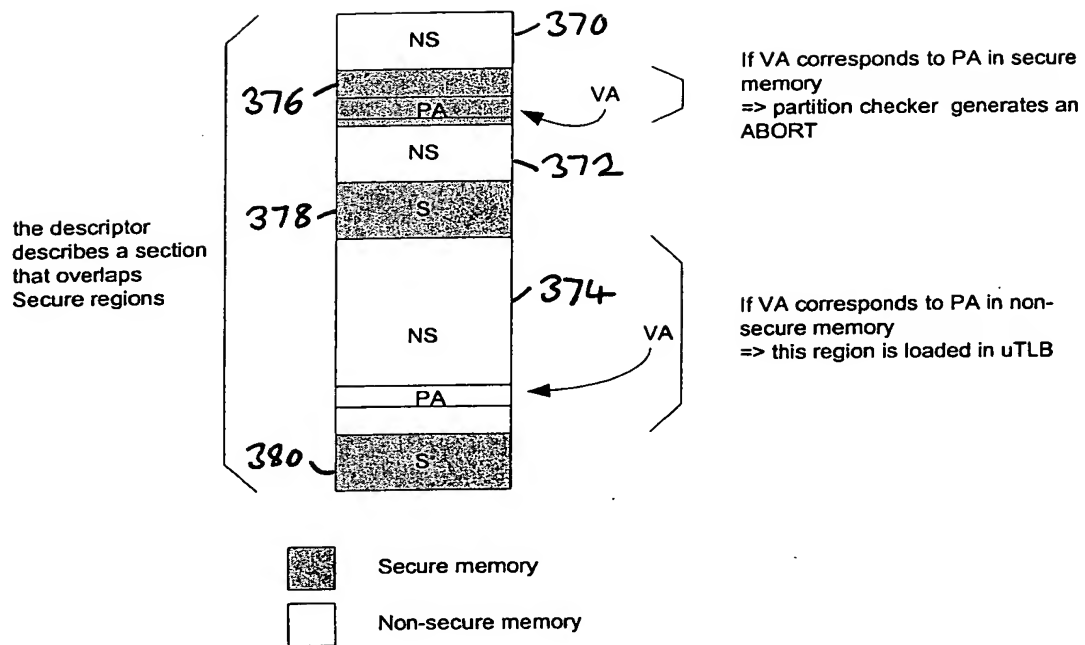


FIG. 41

36/64

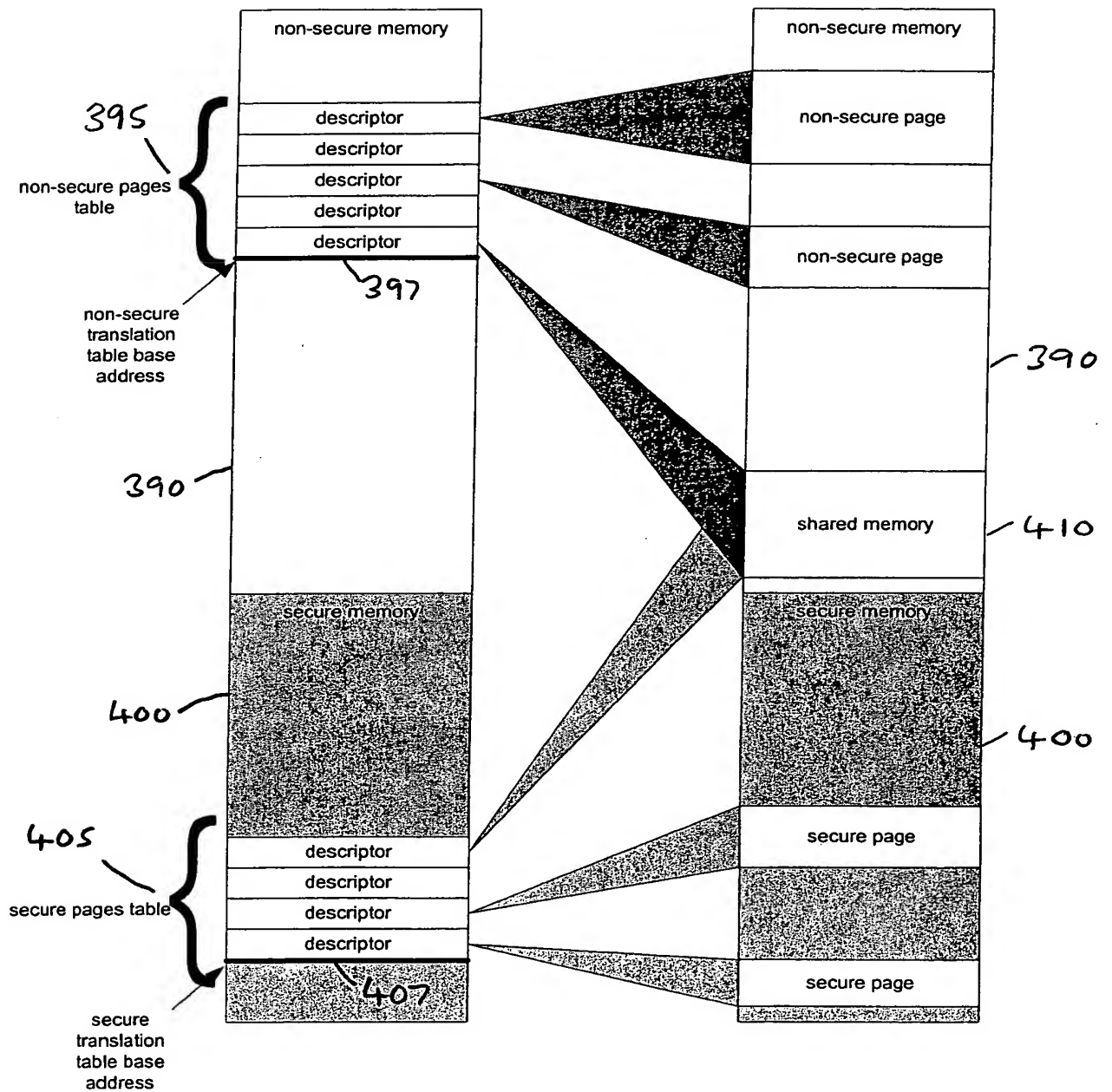


FIG. 42

37/64

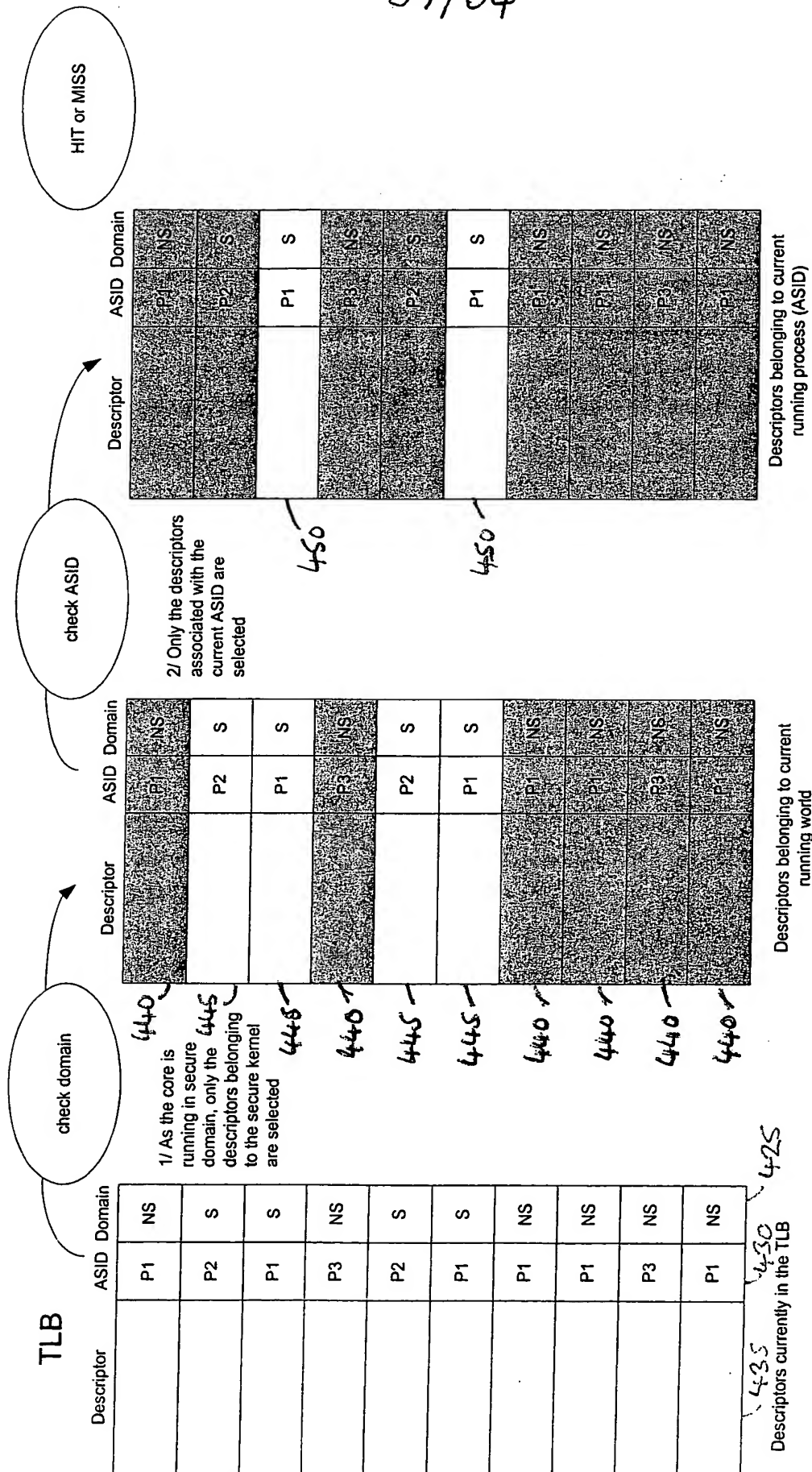


FIG. 43

208

38/64

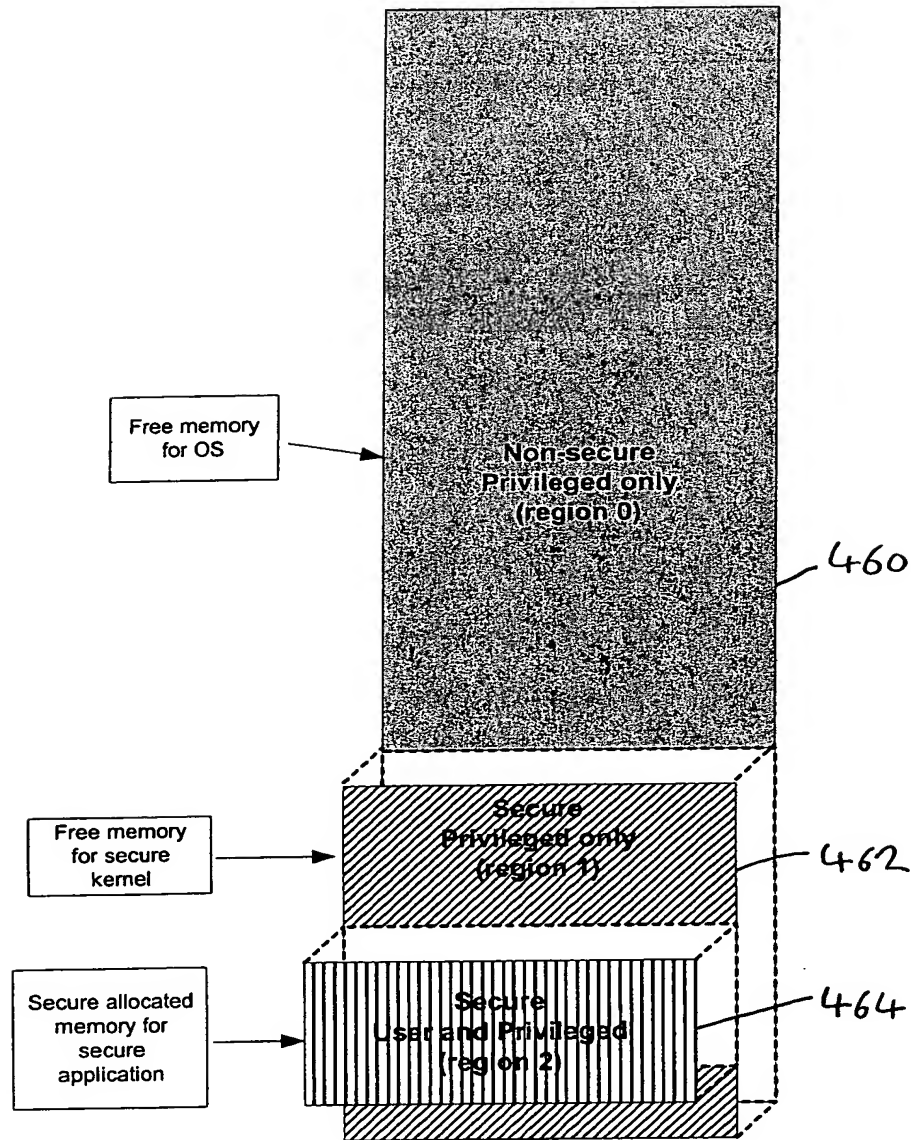


FIG. 44

39/64

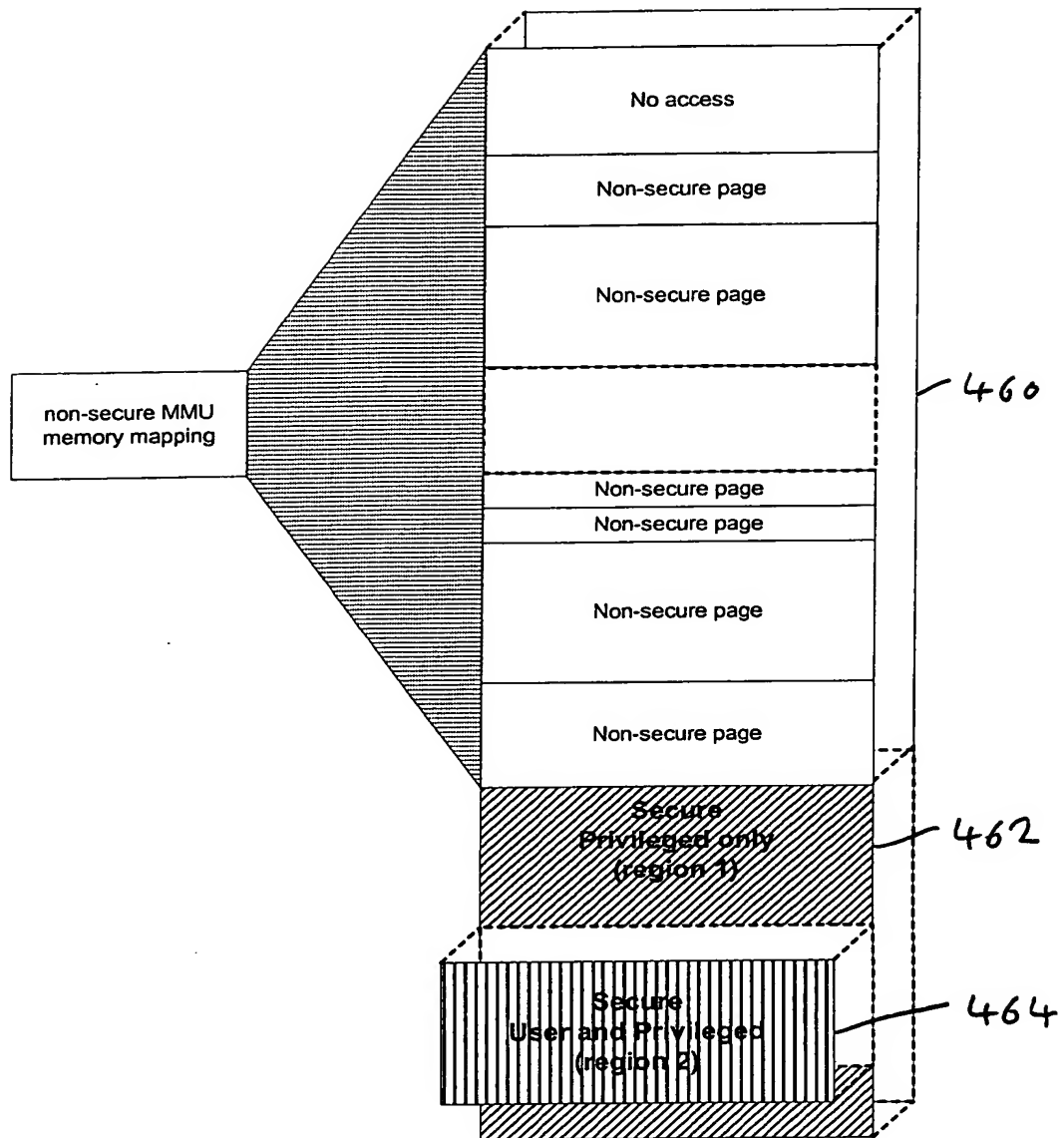


FIG. 45

40/64

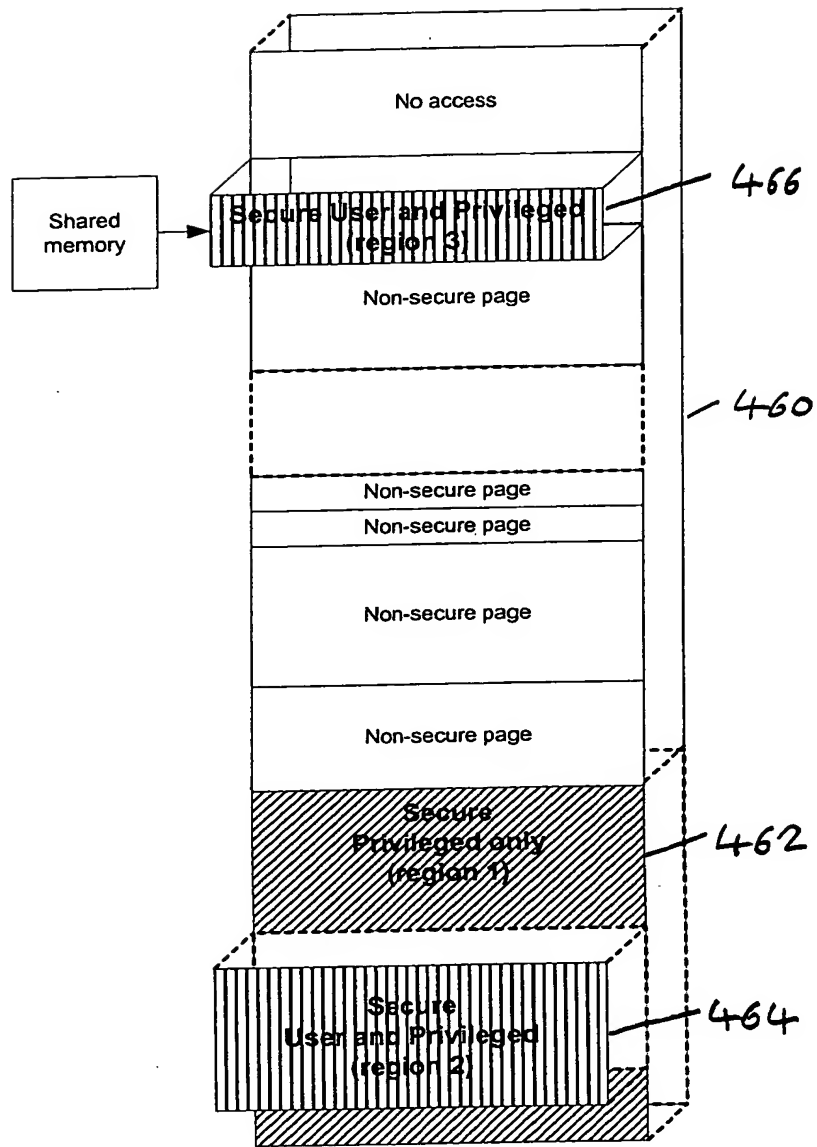


FIG. 46



41/64

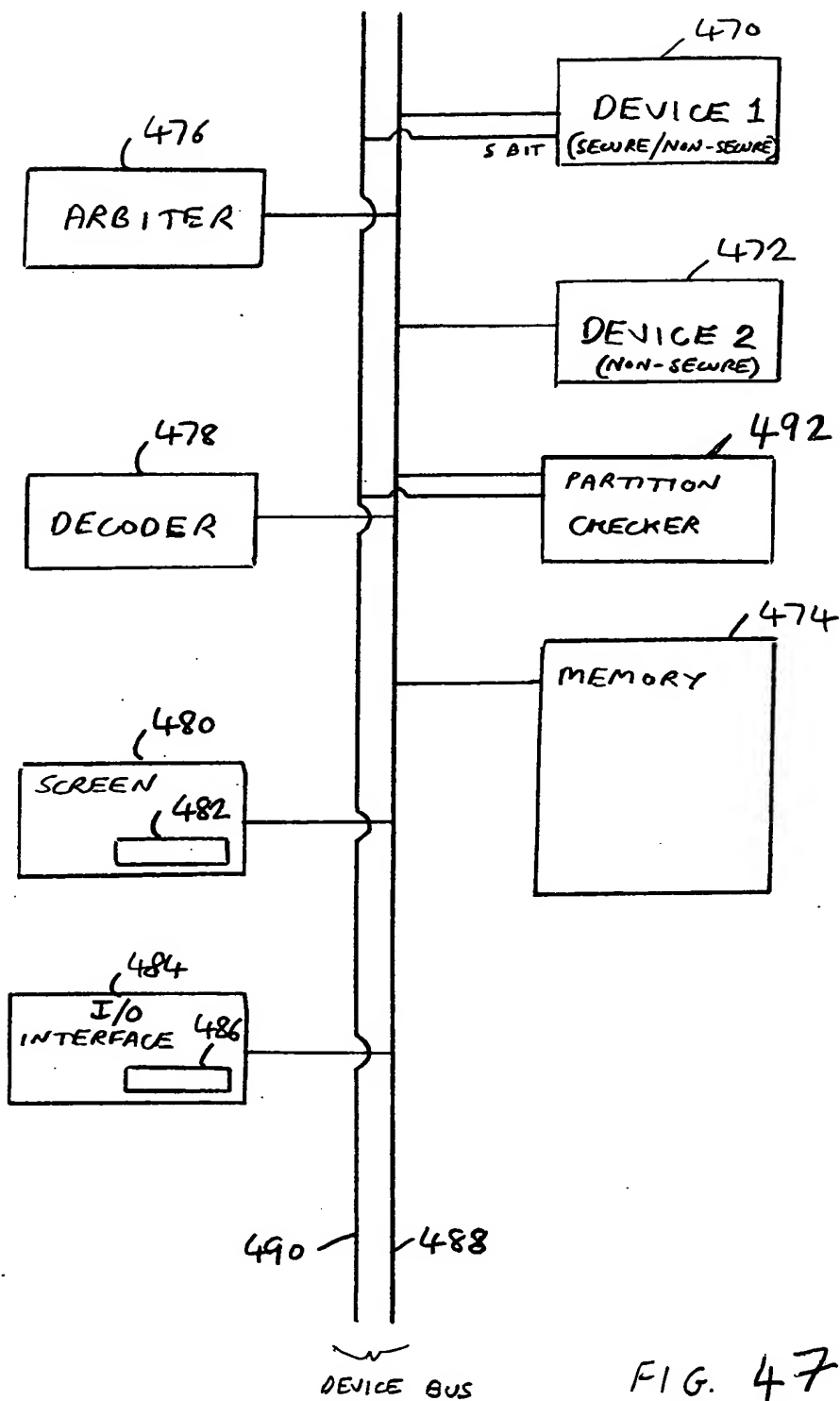


FIG. 47

42/64

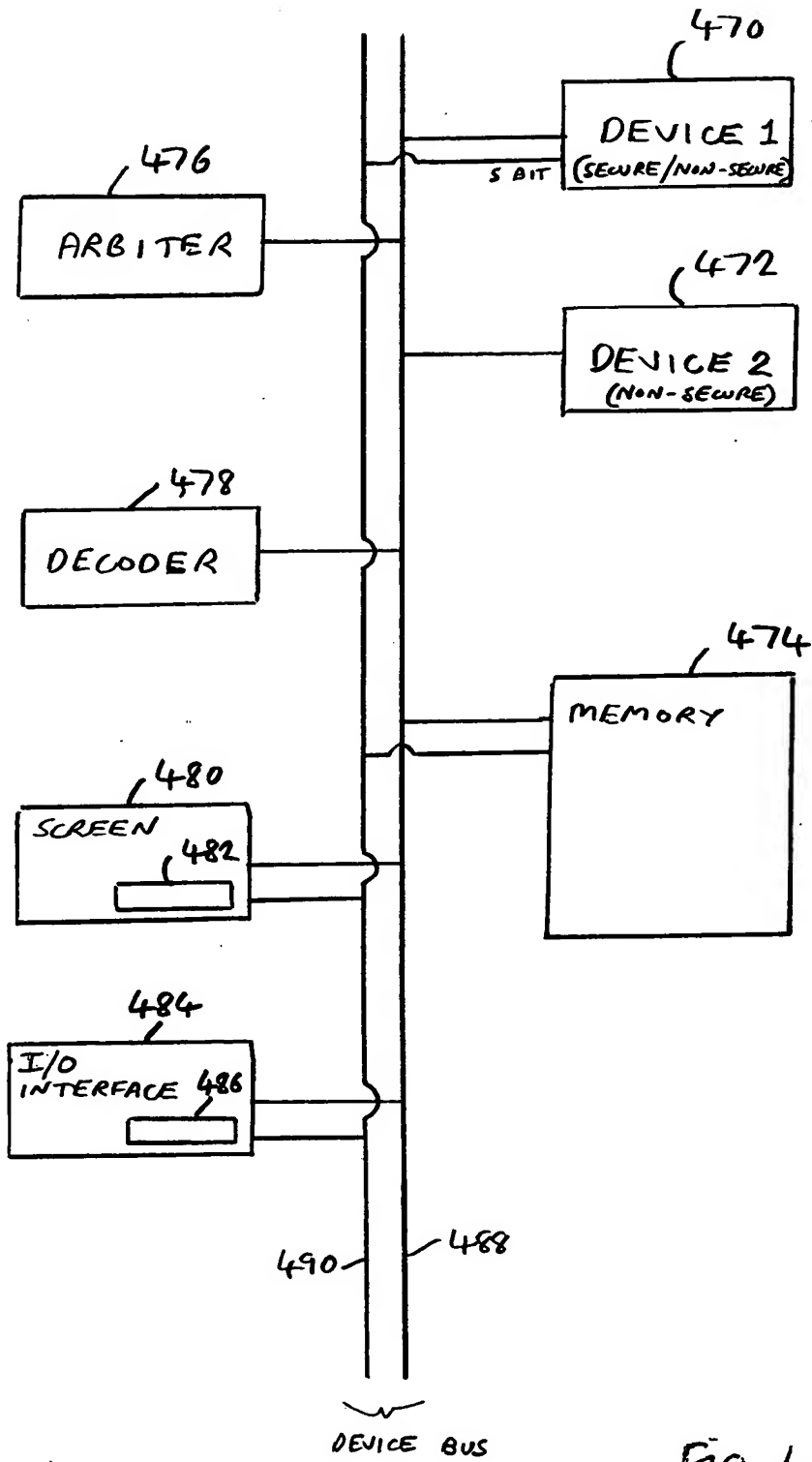


Fig. 48

43/64

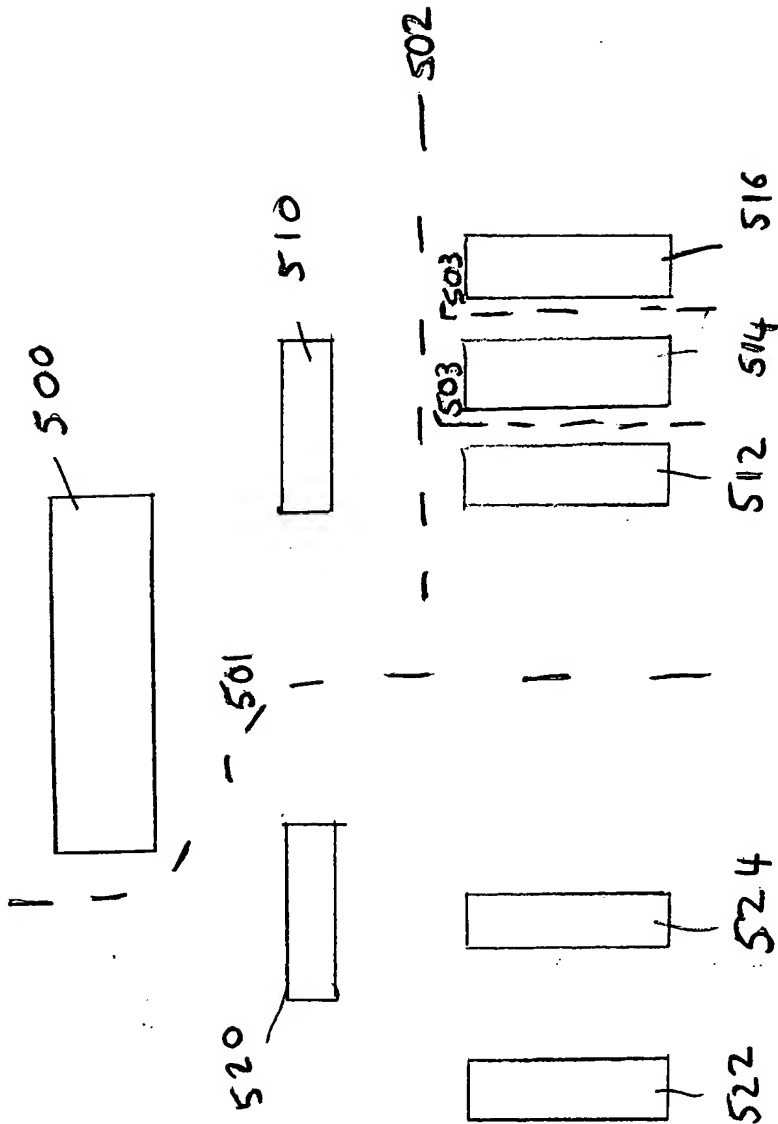
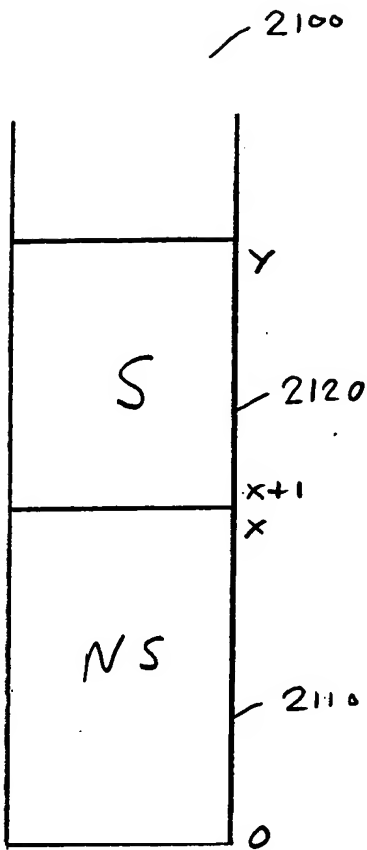


Figure 59

44/64



PHYSICAL  
ADDRESS SPACE

FIG. 49

45/64

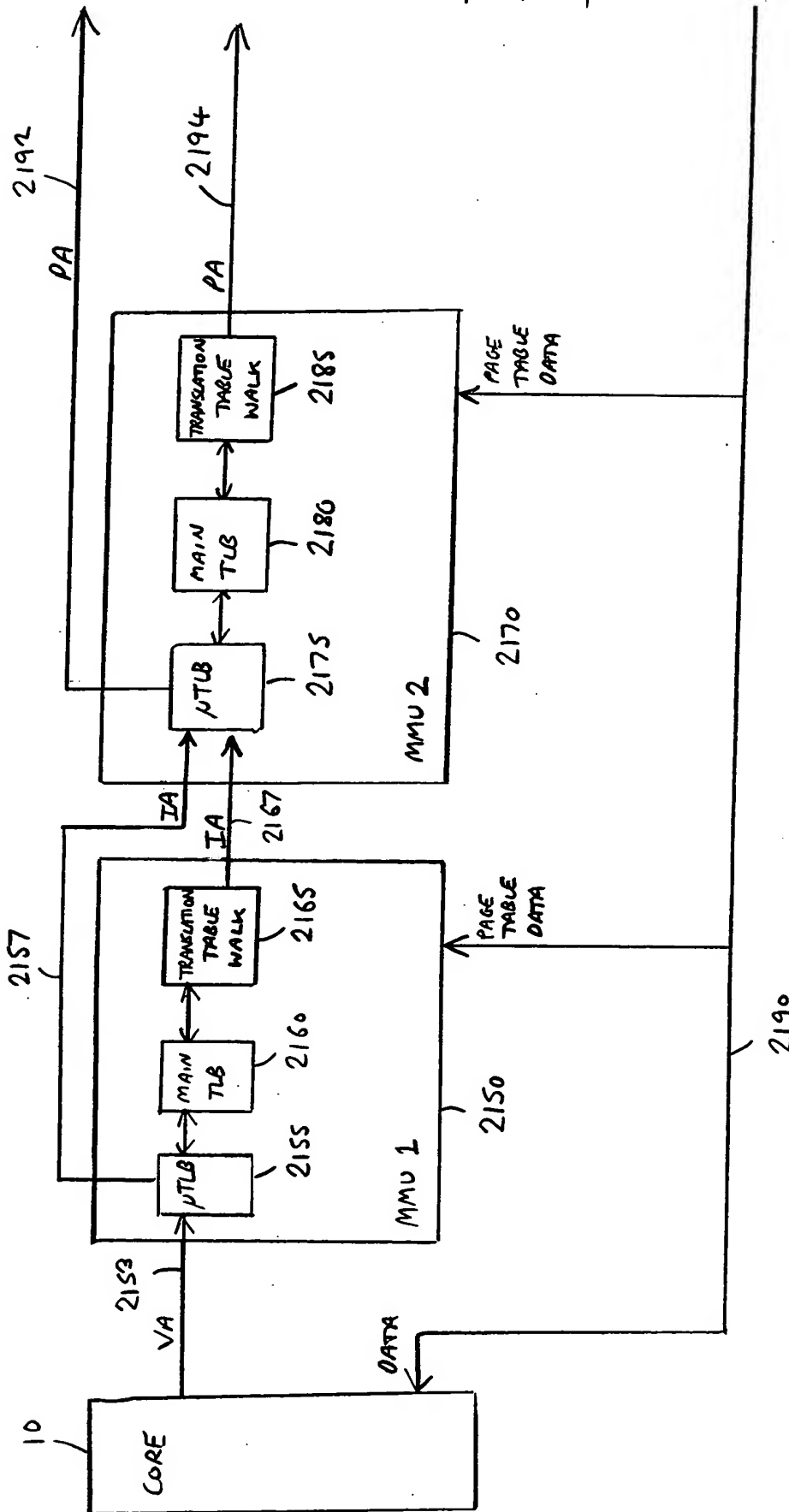


FIG 50A

46/64

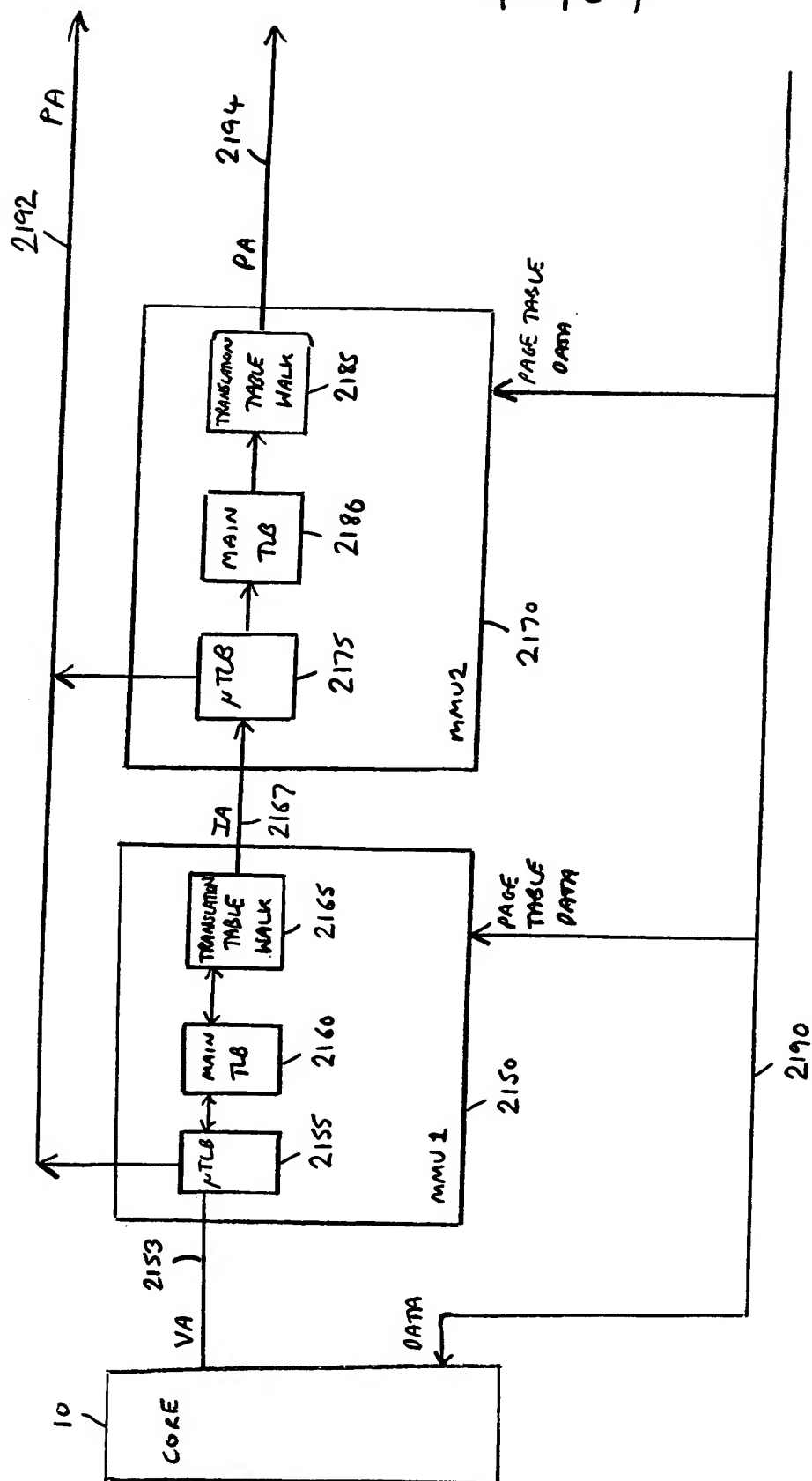


FIG 50B

47/64

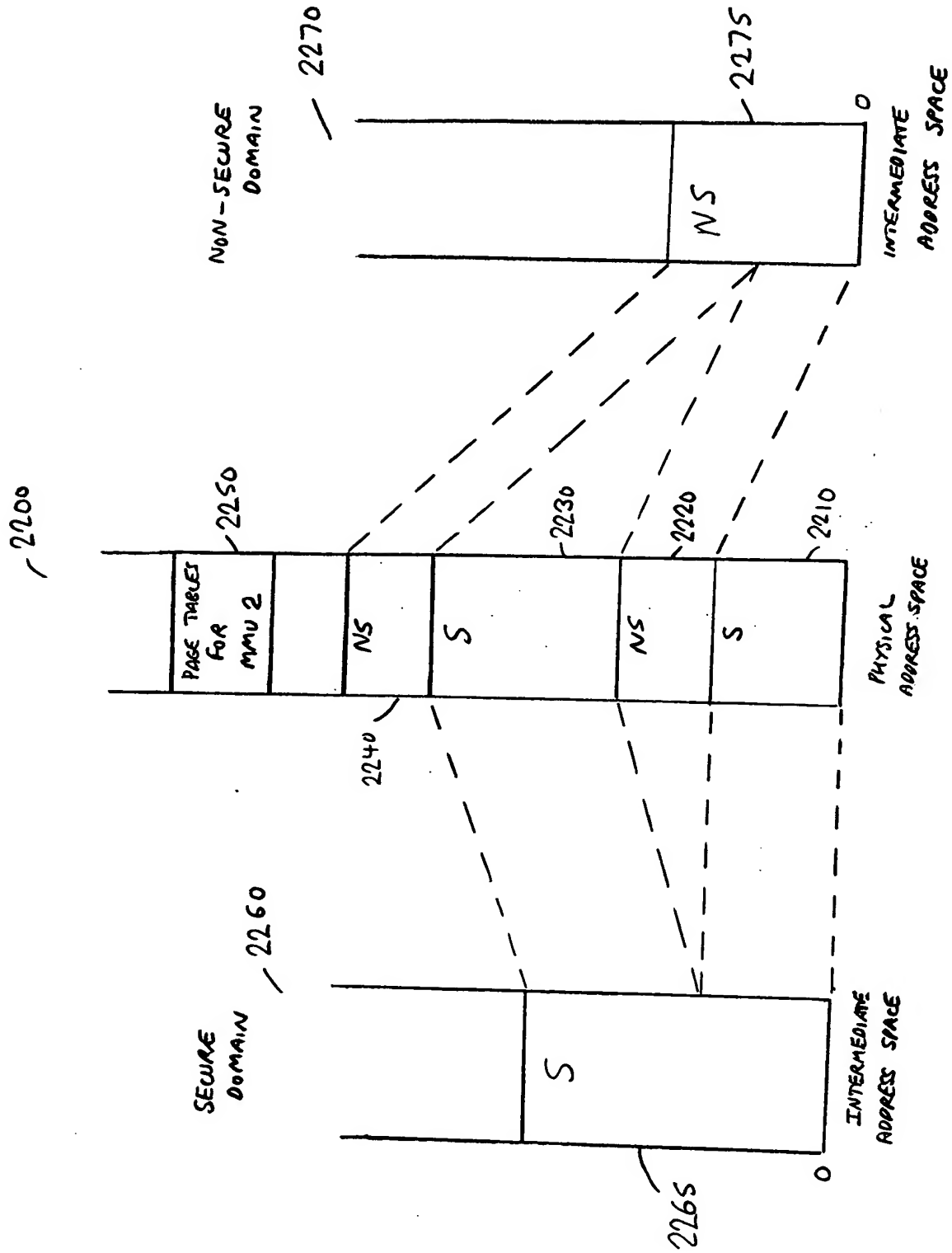


FIG 51

48/64

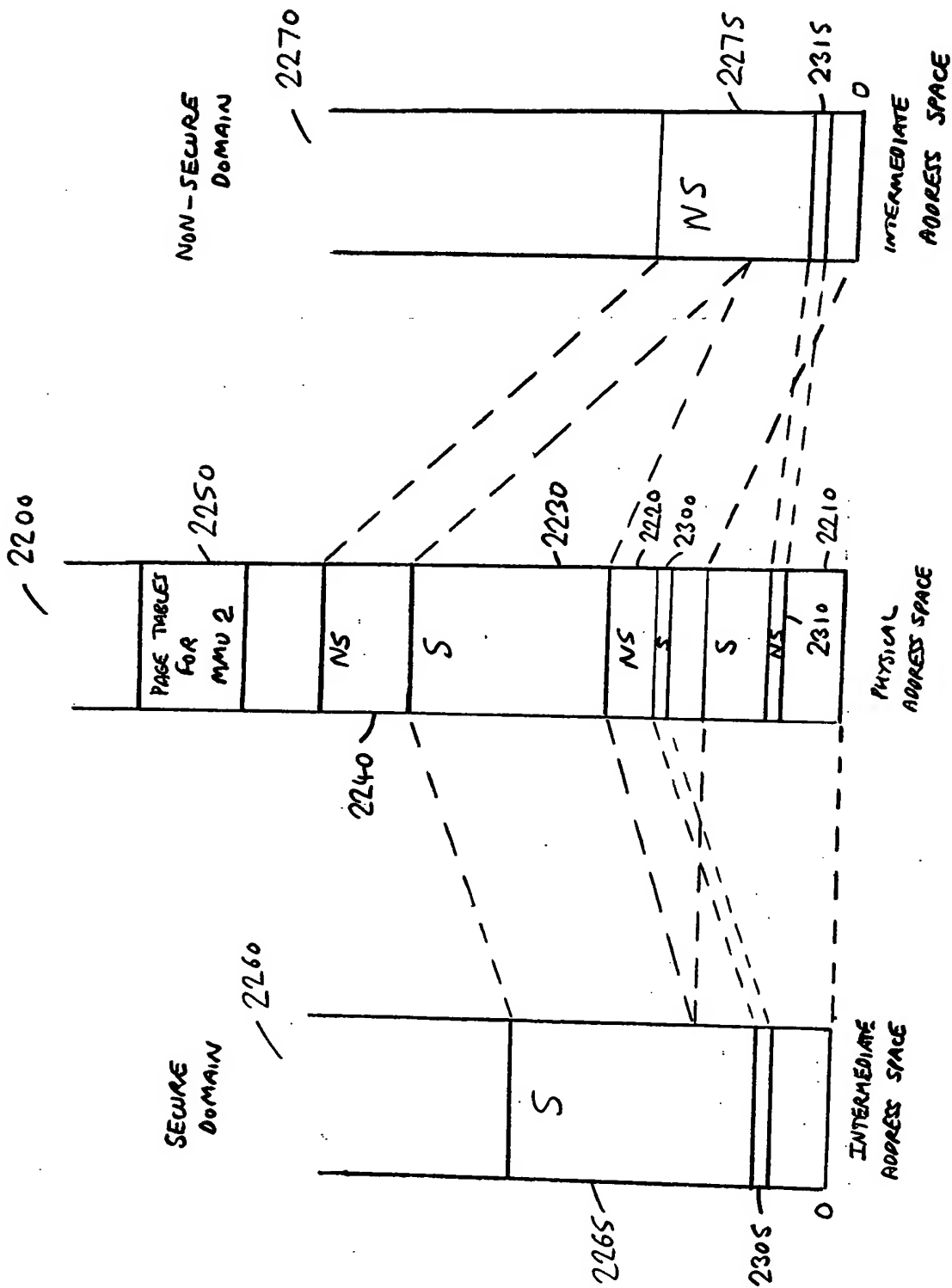


FIG 52



49/64

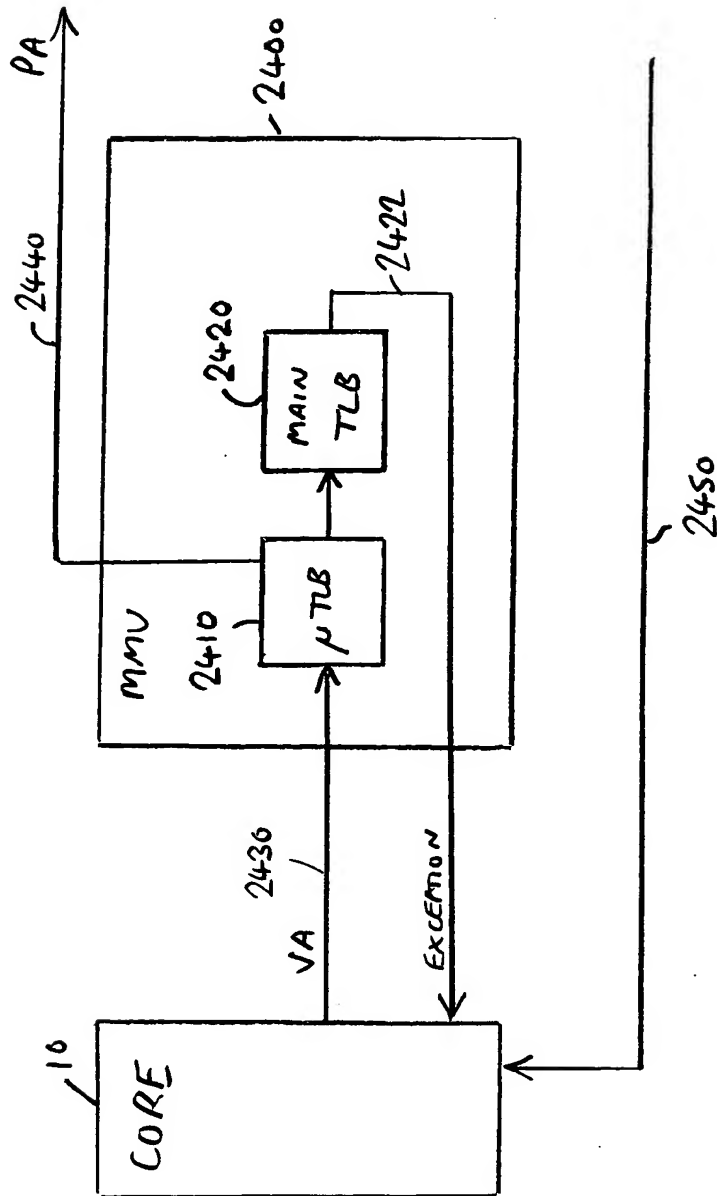


FIG 53

50/64

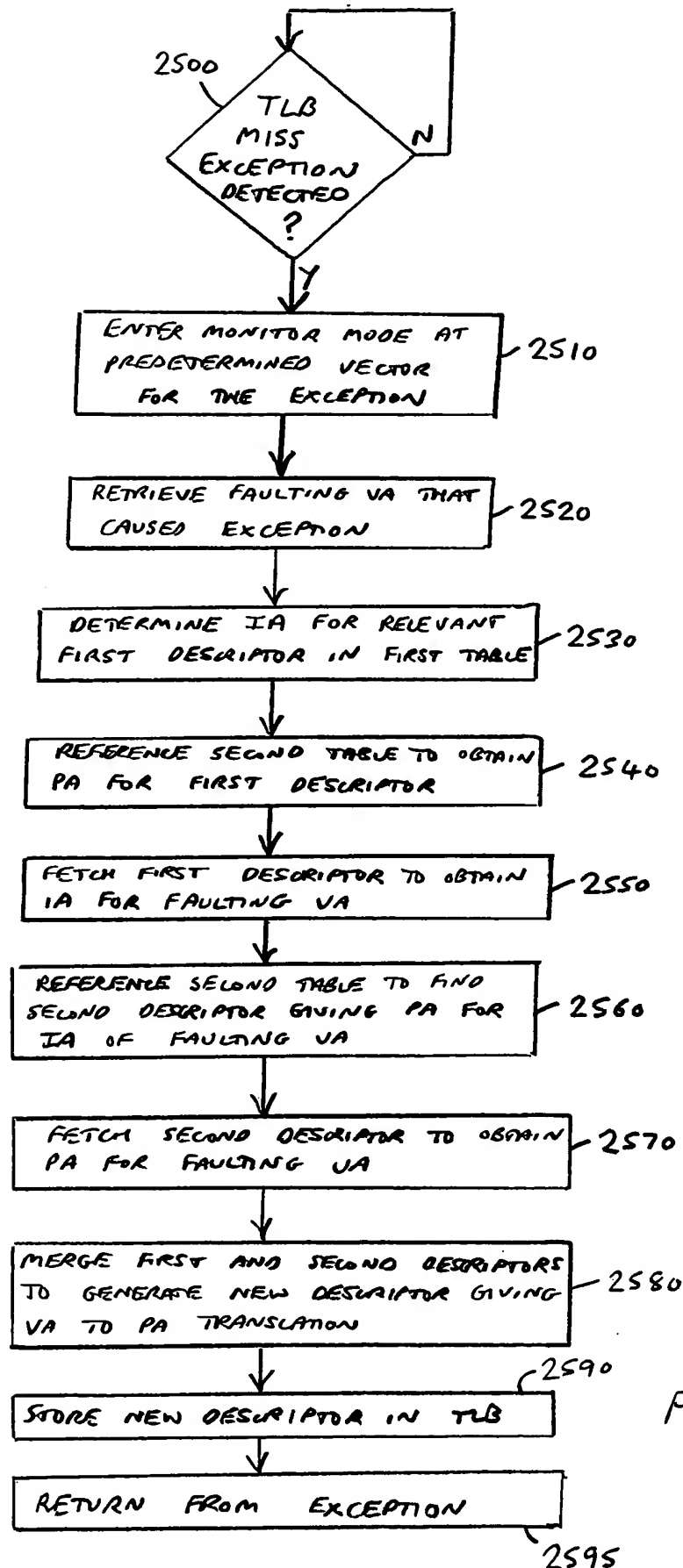


FIG 54

51/64

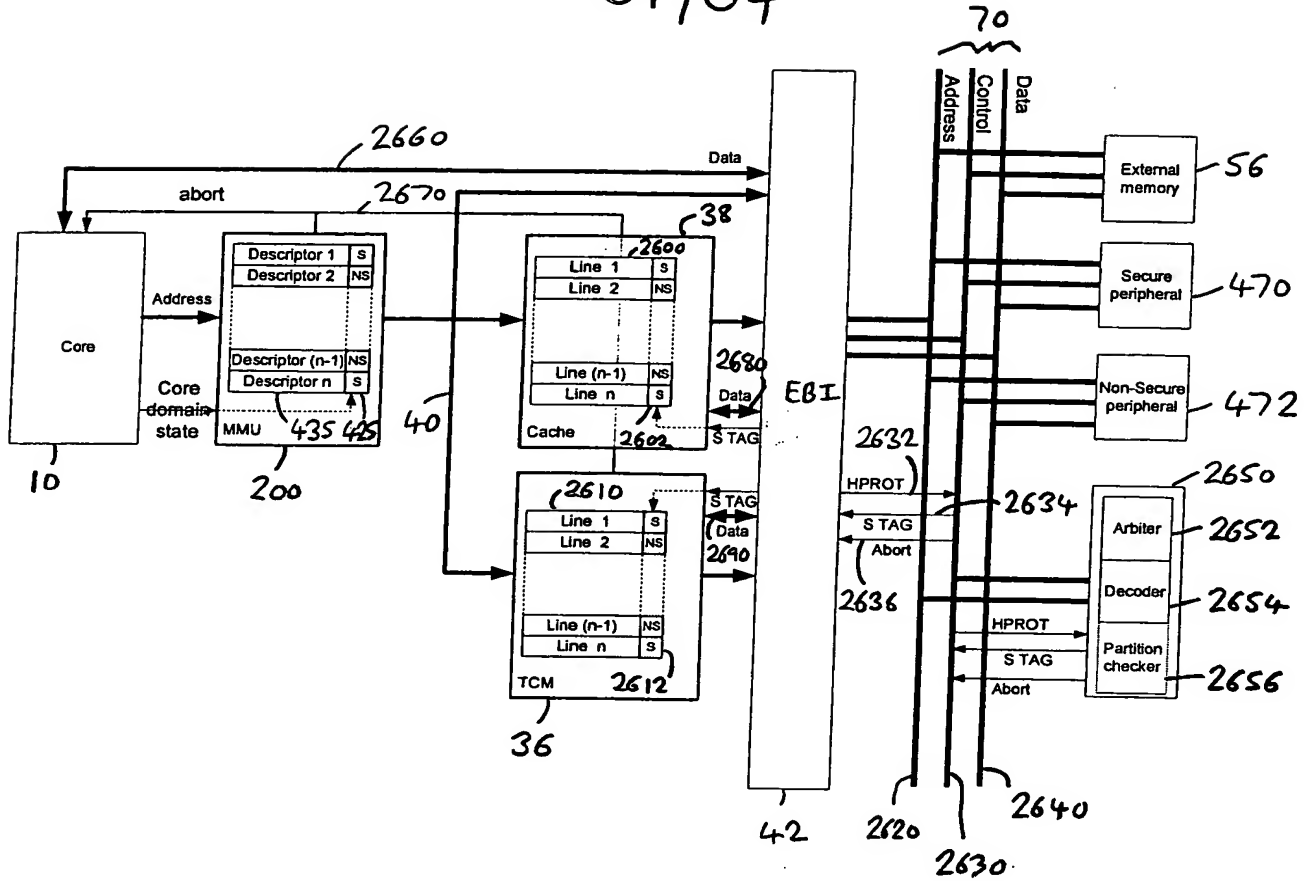


FIG 55

The diagram illustrates a memory management unit (MMU) architecture. A **CORE** (10) is connected to the MMU (200) via a bus. The MMU (200) contains several components:
 

- access permission** (202) and **region attribute** (204) blocks, which are **cachable, bufferable** (232).
- μTLB** (206) and **main TLB** (208) blocks.
- descriptors** (246) and a **translation table walk** block (210).

 The **CORE** (10) sends a **virtual address** (234) to the **μTLB** (206). The **μTLB** (206) outputs to the **main TLB** (208) via a path labeled 242. The **main TLB** (208) outputs to the **descriptors** (246), which then lead to the **translation table walk** block (210). The **translation table walk** block (210) outputs a **physical address** (248). The **CORE** (10) also receives an **abort** signal (230) from the MMU (200) and a **physical address** (236) from the **translation table walk** block (210). A bus labeled 238 connects the **CORE** (10) to the MMU (200).

FIG 56

53/64

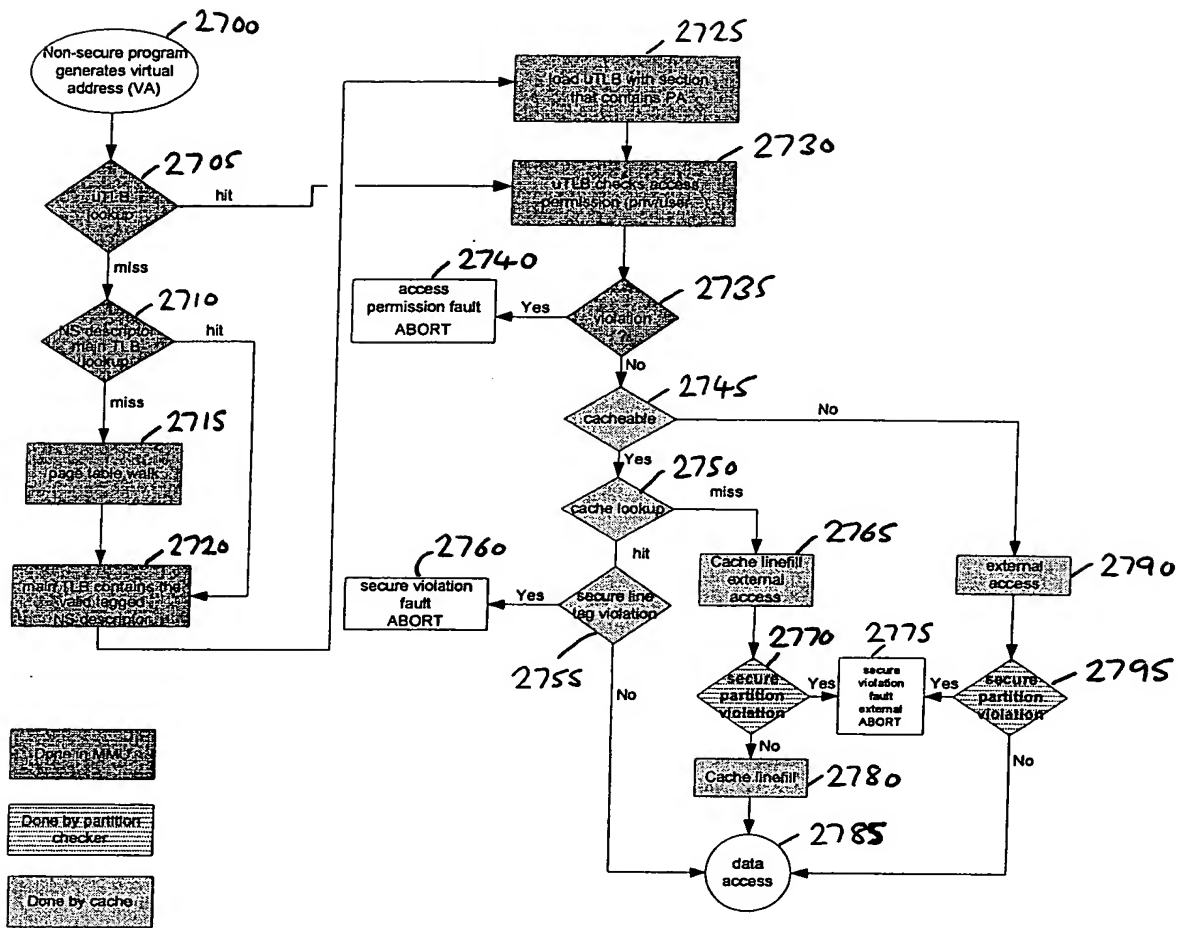


FIG. 57

54/64

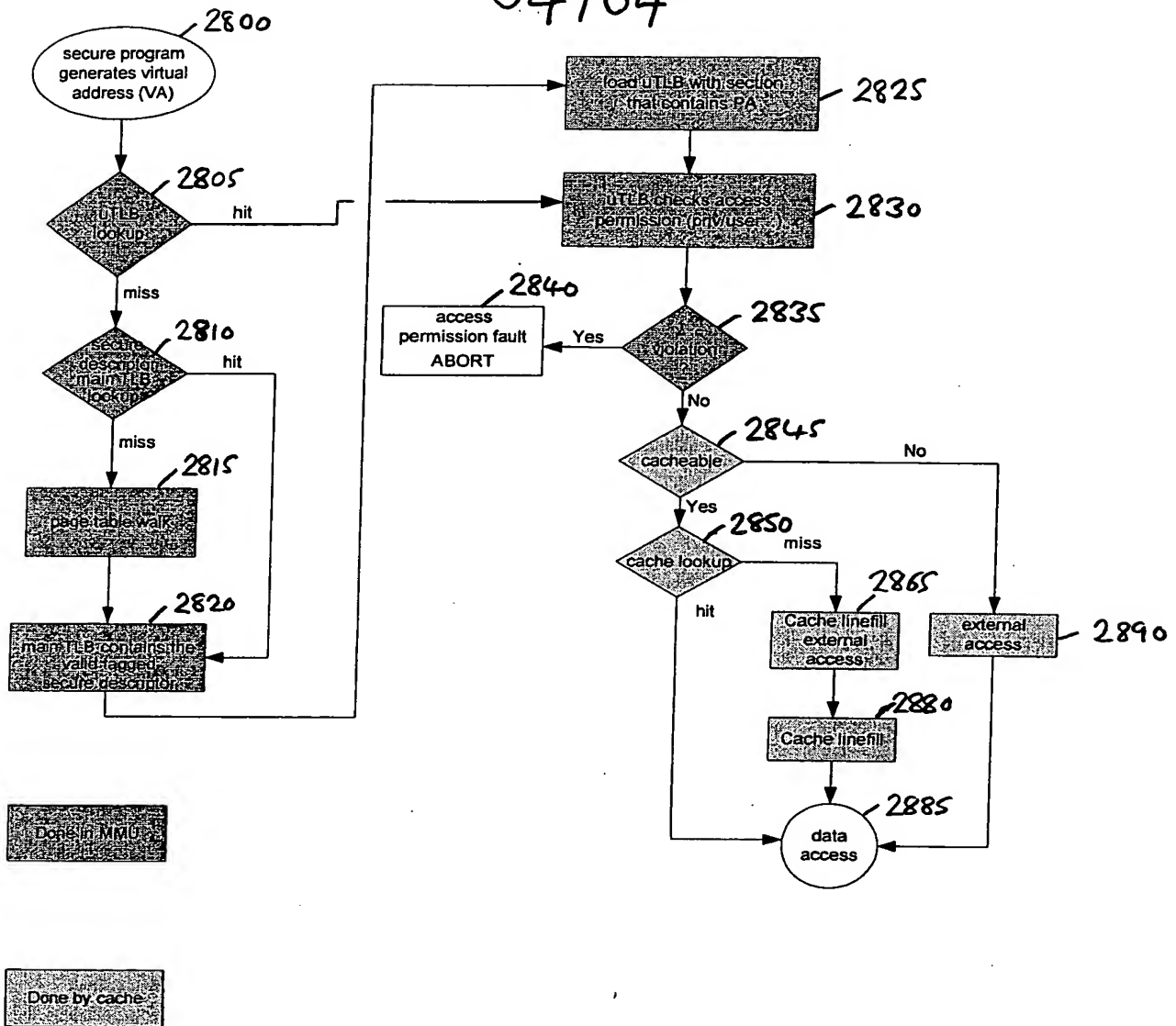


FIG 58

55/64

Method of entry	How to program?	How to enter?	Entry mode
Breakpoint hits	Debug TAP or software (CP14)	Program breakpoint register and/or context-ID register and comparisons succeed with Instruction Address and/or CP15 Context ID <sup>(2)</sup> .	Halt/monitor <sup>(1)</sup>
Software breakpoint instruction	Put a BKPT instruction into scan chain 4 (Instruction Transfer Register) through Debug TAP or Use BKPT instruction directly in the code.	BKPT instruction must reach execution stage.	Halt/monitor
Vector trap breakpoint	Debug TAP	Program vector trap register and address matches.	Halt/monitor
Watchpoint hits	Debug TAP or software (CP14)	Program watchpoint register and/or context-ID register and comparisons succeed with Instruction Address and/or CP15 Context ID <sup>(2)</sup> .	Halt/monitor <sup>(1)</sup>
Internal debug request	Debug TAP	Halt instruction has been scanned in.	Halt
External debug request	Not applicable	EDBGRQ input pin is asserted.	Halt

<sup>(1)</sup>: In monitor mode, breakpoints and watchpoints cannot be data-dependent.

<sup>(2)</sup>: The cores have support for thread-aware breakpoints and watchpoints, in order to enable secure debug on some particular threads.

Figure 60

56/64




Name	Meaning	Reset value	Access	Inserted in scan chain for test
Monitor mode enable bit	0: halt mode 1: monitor mode	1	R/W by programming the ICE by the JTAG (scan1) ▪ R/W by using MRC/MCR instruction (CP14)	yes
Secure debug enable bit	0: debug in non-secure world only. 1: debug in secure world and non-secure world	0	In functional mode or debug monitor mode: R/W by using MRC/MCR instruction (CP14) (only in secure supervisor mode)  In Debug halt mode: No access – MCR/MRC instructions have any effect.  (R/W by programming the ICE by the JTAG (scan1) if JSDAEN=1	no
Secure trace enable bit	0: ETM is enabled in non-secure world only. 1: ETM is enabled in secure world and non-secure world	0	In functional mode or debug monitor mode: R/W by using MRC/MCR instruction (CP14) (only in secure supervisor mode)  In Debug halt mode: No access – MCR/MRC instructions have any effect.  (R/W by programming the ICE by the JTAG (scan1) if JSDAEN=1	no
Secure user-mode enable bit	0: debug is not possible in secure user mode 1: debug is possible in secure user mode	1	In functional mode or debug monitor mode: R/W by using MRC/MCR instruction (CP14) (only in secure supervisor mode)  In Debug halt mode: No access – MCR/MRC instructions have any effect.  (R/W by programming the ICE by the JTAG (scan1) if JSDAEN=1	no
Secure thread-aware enable bit	0: debug is not possible for a particular thread 1: debug is possible for a particular thread	0	In functional mode or debug monitor mode: R/W by using MRC/MCR instruction (CP14) (only in secure supervisor mode)  In Debug halt mode: No access – MCR/MRC instructions have any effect.  (R/W by programming the ICE by the JTAG (scan1) if JSDAEN=1	no

Figure 6/



57/64

**Function Table**

D	CK	Q[n+1]
0		0
1		1
X		Q[n]

**Logic Symbol**

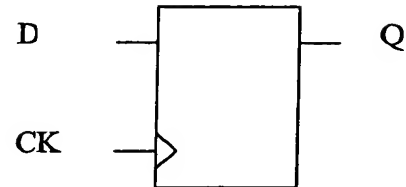







FIGURE 62

**Function Table**

D	SI	SE	CK	Q[n+1]
0	X	0		0
1	X	0		1
X	X	X		Q[n]
X	0	1		0
X	1	1		1

**Logic Symbol**

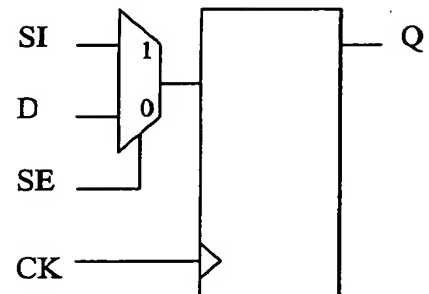


figure 63

58/64

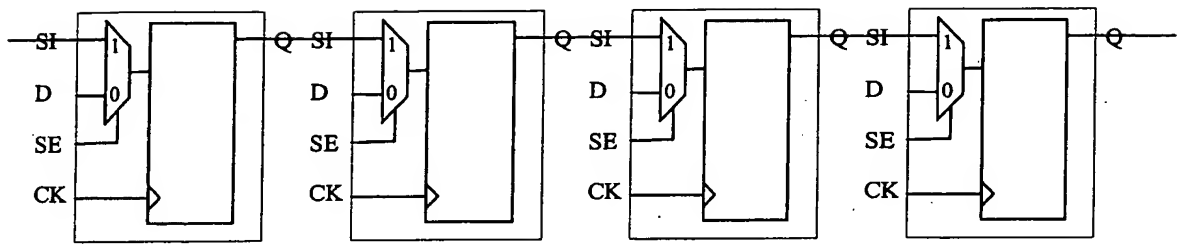


FIGURE 64

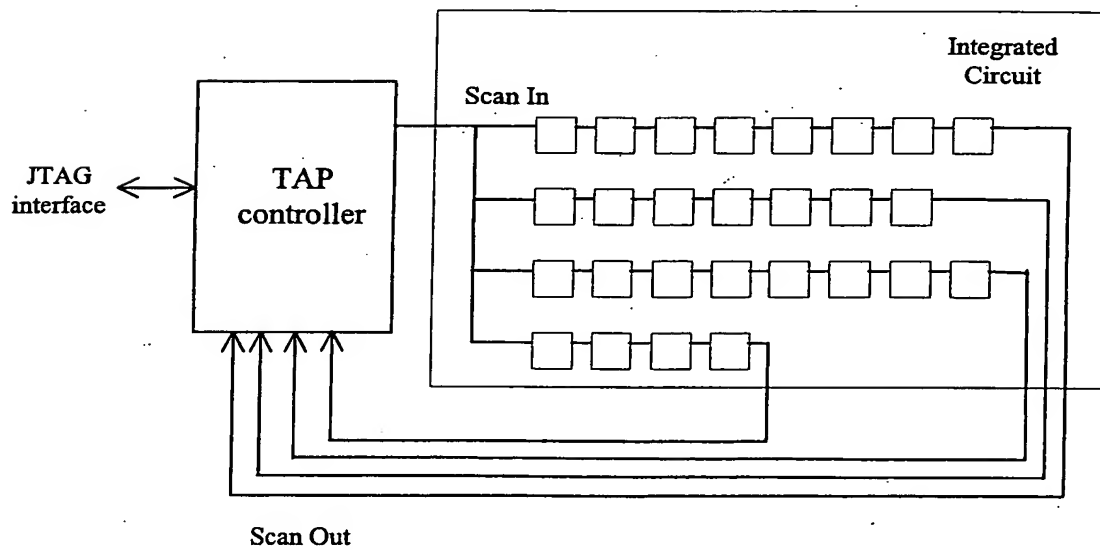


Figure 65.

59/64

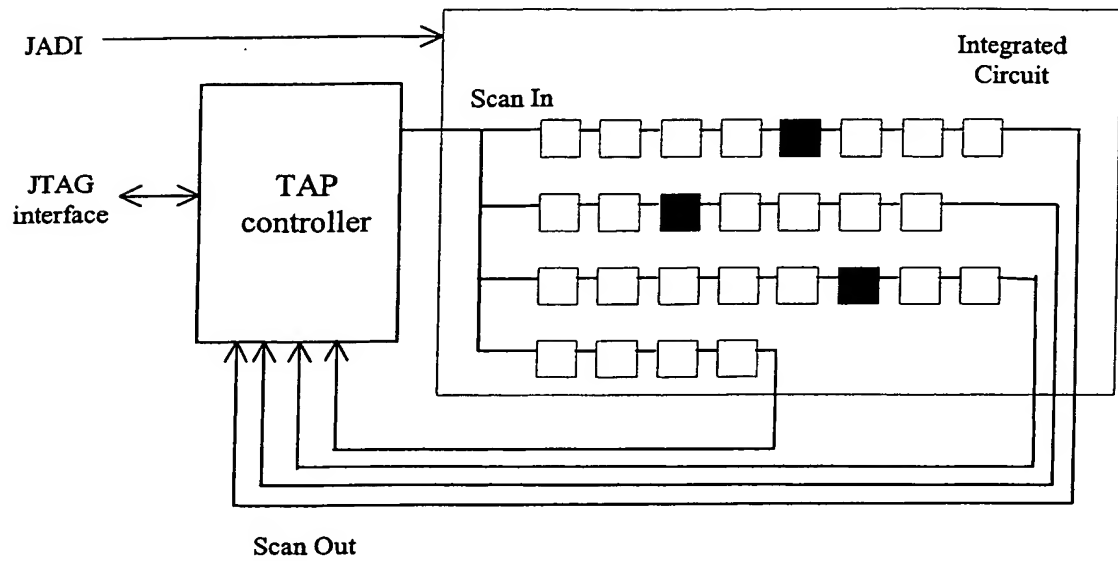


FIGURE 66A

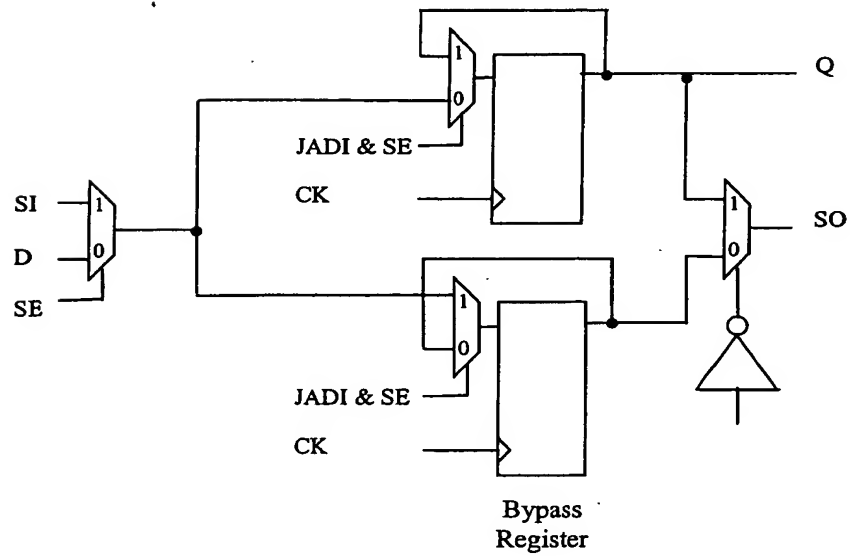


FIGURE 66 B

60/64

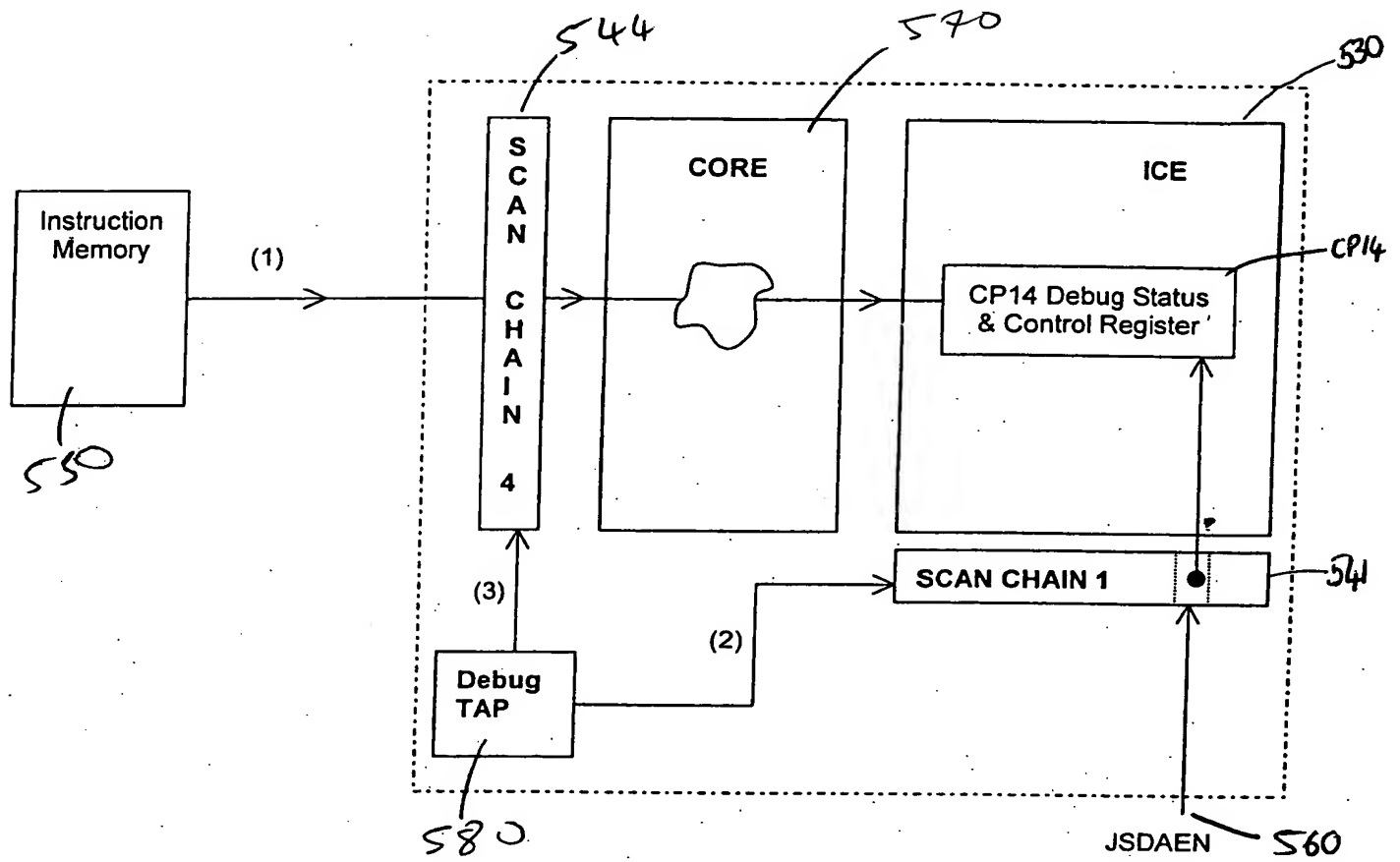


Figure 67

61/64

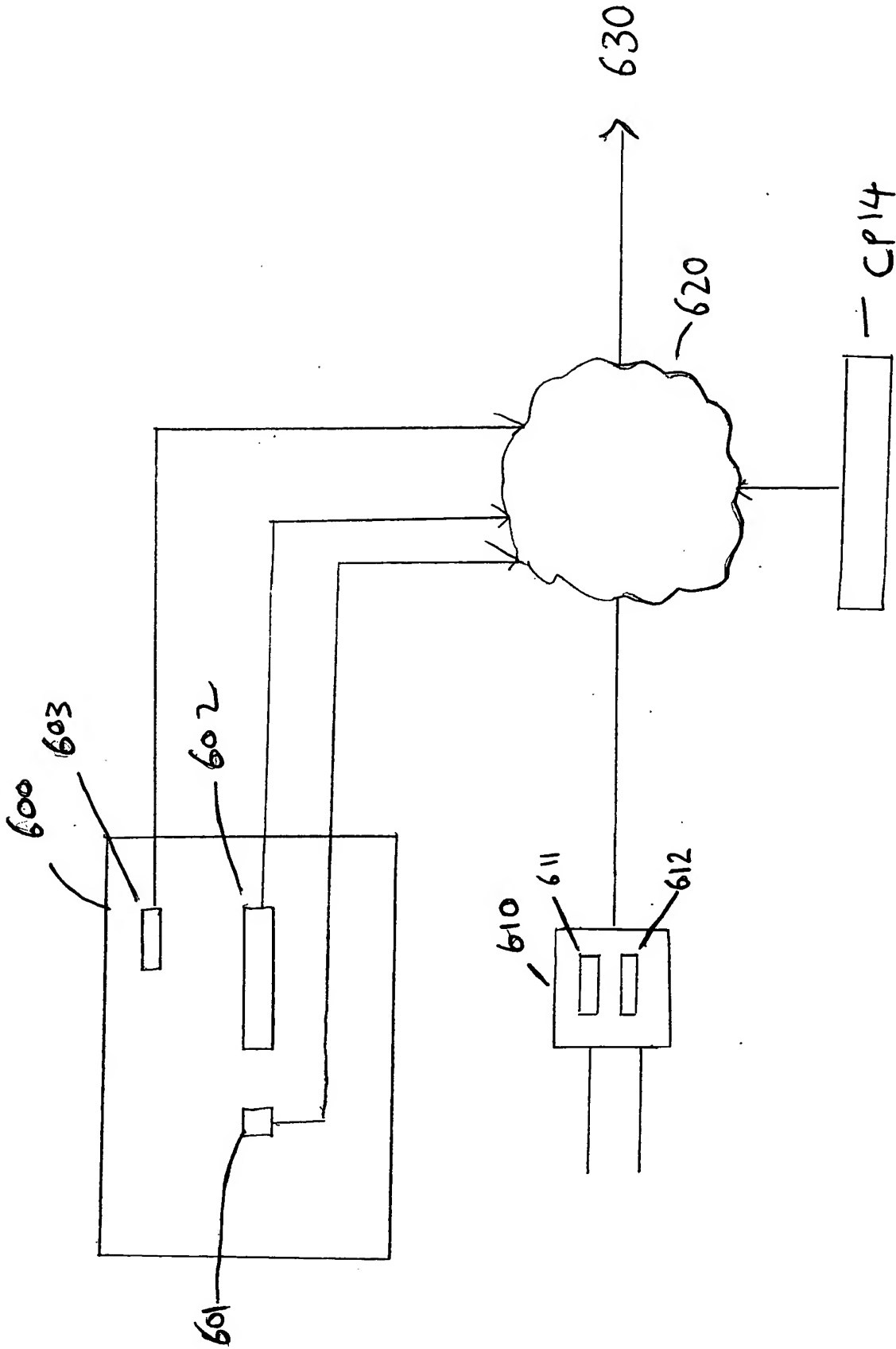


Figure 68

62/64

CP14 bits in Debug and Status Control register			meaning
Secure debug enable bit	Secure user-mode debug enable bit	Secure thread-aware debug enable bit	
0	X	X	No intrusive debug in entire secure world is possible. Any debug request, breakpoints, watchpoints, and other mechanism to enter debug state are ignored in entire secure world.
1	0	X	Debug in entire secure world is possible
1	1	0	Debug in secure user-mode only. Any debug request, breakpoints, watchpoints, and other mechanism to enter debug state are taken into account in user mode only. (Breakpoints and watchpoints linked or not to a thread ID are taken into account). Access in debug is restricted to what secure user can have access to.
1	1	1	Debug is possible only in some particular threads. In that case only thread-aware breakpoints and watchpoints linked to a thread ID are taken into account to enter debug state. Each thread can moreover debug its own code, and only its own code.

Figure 69A

CP14 bits in Debug and Status Control register			meaning
Secure trace enable bit	Secure user-mode debug enable bit	Secure thread-aware debug enable bit	
0	X	X	No observable debug in entire secure world is possible. Trace module (ETM) must not trace internal core activity.
1	0	X	Trace in entire secure world is possible
1	1	0	Trace is possible when the core is in secure user-mode only.
1	1	1	Trace is possible only when the core is executing some particular threads in secure user mode. Particular hardware must be dedicated for this, or re-use breakpoint register pair: Context ID match must enable trace instead of entering debug state.

Figure 69B

63/64

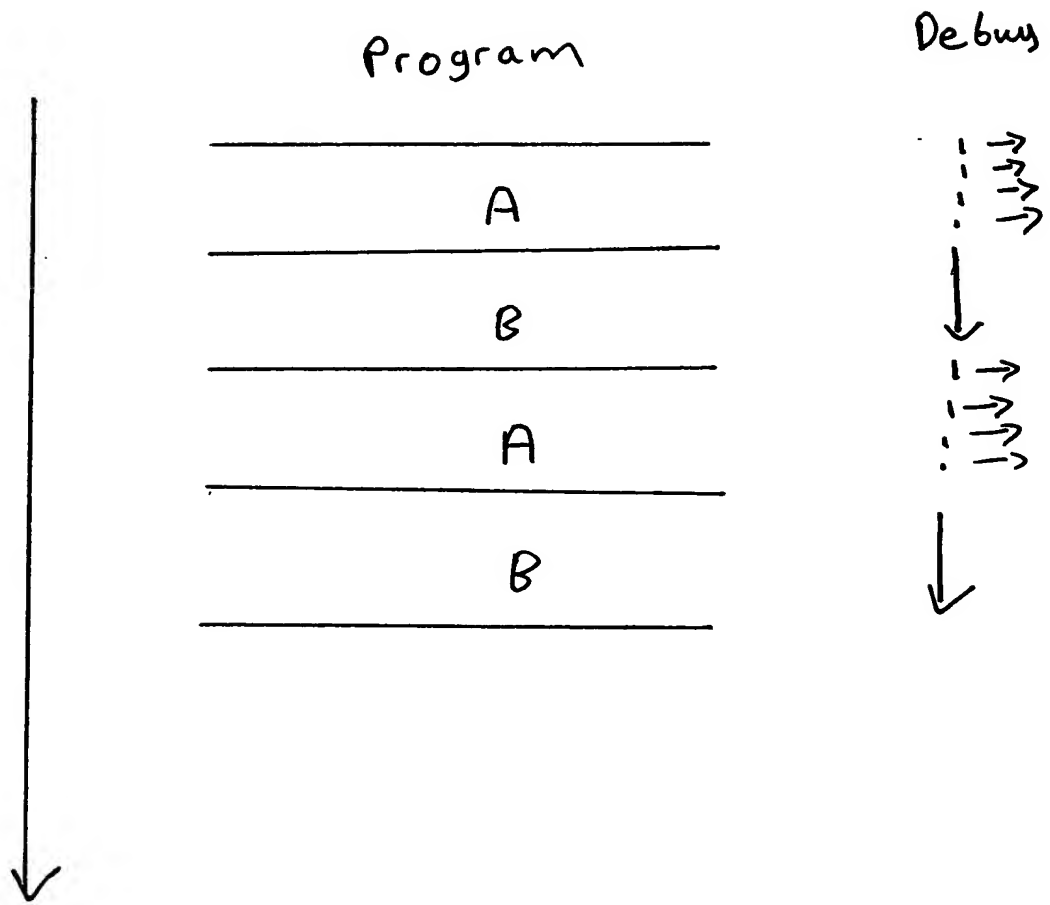


Figure 70

64/64

Method of entry	Entry when in non-secure world	entry when in secure world
Breakpoint hits	Non-secure prefetch abort handler	secure prefetch abort handler
Software breakpoint instruction	Non-secure prefetch abort handler	secure prefetch abort handler
Vector trap breakpoint	Disabled for non-secure data abort and non-secure prefetch abort interruptions. For other non-secure exceptions, prefetch abort.	Disabled for secure data abort and secure prefetch abort exceptions <sup>(1)</sup> . For other exceptions, secure prefetch abort.
Watchpoint hits	Non-secure data abort handler	secure data abort handler
Internal debug request	Debug state in halt mode	debug state in halt mode
External debug request	Debug state in halt mode	debug state in halt mode

(1) see information on vector trap register, :

(2) Note that when external or internal debug request is asserted, the core enters halt mode and not monitor mode.

**Figure 71A**

Method of entry	Entry in non-secure world	entry in secure world
Breakpoint hits	Non-secure prefetch abort handler	breakpoint ignored
Software breakpoint instruction	Non-secure prefetch abort handler	instruction ignored <sup>(1)</sup>
Vector trap breakpoint	Disabled for non-secure data abort and non-secure prefetch abort interruptions. For others interruption non-secure prefetch abort.	breakpoint ignored
Watchpoint hits	Non-secure data abort handler	watchpoint ignored
Internal debug request	Debug state in halt mode	request ignored
External debug request	Debug state in halt mode	request ignored
Debug re-entry from system speed access	not applicable	not applicable

<sup>(1)</sup> As substitution of BKPT instruction in secure world from non-secure world is not possible, non-secure abort must handle the violation.

**Figure 71B**